TAMPEREEN
AMMATTIKORKEAKOULU

PELIEDITORIN TUOMINEN OSAKSI OPETUSTA

Planetoid Pioneers -aloitusoppaan laatiminen

Veera Tikkamäki

Opinnäytetyö
Marraskuu 2018
Tietojenkäsittely
Pelituotanto

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietojenkäsittely
Pelituotanto

TIKKAMÄKI, VEERA:
Pelieditorin tuominen osaksi opetusta
Planetoid Pioneers -aloitusoppaan laatiminen

Opinnäytetyö 89 sivua, joista liitteitä 48 sivua
Marraskuu 2018

Videopeliteollisuus on ollut kasvava ala jo pitkään, eikä suunta ole lähivuosina muuttumassa. Suosio näkyy myös kouluissa: on olemassa useita tapoja, joilla pelien synnyttämää intoa on pyritty tuomaan luokkahuoneisiin. Yksi näistä on pelinteko, jonka ajankohtaisuuden puolesta puhuu pelialan suuruuden lisäksi nykyaikana vallitseva itsetekemisen kulttuuri. Opinnäytetyö tehtiin toimeksiantona 5 More Minutes -opetuspeliyritykselle. Yrityksen tärkein tuote on TeacherGaming Desk -alusta, jonka kautta on saatavilla monia opetuskäyttöön soveltuvia pelejä. Yksi niistä on Planetoid Pioneers -pelipelieditori. Sitä ei ole suunniteltu opetuskäyttöön ja koska kyseessä on monimutkainen ohjelma, on kynnys sen käyttöönottamiselle opetustilanteessa korkea. Työn tavoite oli tämän kynnyksen madaltaminen luomalla Planetoid Pioneersille kattava aloitusopas.

Työn teoriaosuudessa tutkittiin ensin pelien opetuskäytön yleistä teoriaa, minkä jälkeen tarkennettiin pelintekoon opetuksen osana. Lopulta arvioitiin Planetoid Pioneersin vahvuuksia ja heikkouksia opetuskäytössä. Lähteinä käytettiin alan artikkeleita, kirjoja sekä Planetoid Pioneersia käsitteleviä verkkolähteitä. Käytännön osuuden tuloksena syntyi 48-sivuinen, yläasteikäisille oppilaille suunnattu PDF-opas, johon kerättiin materiaali verkossa olevien tutoriaalien avulla. Ohjeistus kirjoitettiin englanniksi toimeksiantajayrityksen kansainvälisen asiakaskunnan vuoksi.

Sekä pelaaminen että pelien tekeminen voivat olla tärkeässä asemassa tulevaisuuden opetuksessa. Itse tekemisen merkitys on suuri, joten Planetoid Pioneersin kaltainen editori voisi olla tärkeä osa 5 More Minutesin tarjontaa. Käytännön työn tuloksena syntynyttä ohjeistusta voitaisiin jatkokehittää testaamalla ja tekemällä muutoksia tämän perusteella. Laajan PDF-dokumentin rinnalle voitaisiin luoda editorin sisältöä suppeammin esittelevää materiaalia. Voitaisiin myös tehdä kokonaan uusi, nykyistä täydentävä ohjeistus. Teoriaosuutta voitaisiin laajentaa tutkimalla julkaisuja, jotka lähestyvät aihetta useammasta eri näkökulmasta.

Asiasanat: pelinteko opetuksessa, konstruktionismi, videopelit opetuksessa, opetuspelit

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in Business Information Systems
Option of Game Development

TIKKAMÄKI, VEERA:
Bringing a Game Editor into Classrooms
Compiling a "Getting started" guide for Planetoid Pioneers

Bachelor's thesis 89 pages, appendices 48 pages
November 2018

This thesis was commissioned by 5 More Minutes, a game company, which focuses on distributing games fit for education. They include a game and game editor Planetoid Pioneers. The game was not designed for classrooms and the threshold for taking it up in schools is high. The aim of this thesis was to make it more accessible by compiling a guide to be distributed with it.

The thesis begins with theory discussing the field of games and learning, first on a general level, then focusing on making games for learning. Eventually, the pros and cons of Planetoid Pioneers were assessed. Articles, books, and web pages were used as sources. The material for the guide was compiled by exploring the editor with the help of existing tutorials. The project resulted in a 48-page instructional PDF document.

Theory seems to indicate that there are potential in both, playing and making games for learning. Therefore, a game editor could be a noteworthy part of 5 More Minutes' selection. In the future, the guide could be further improved, another more advanced one could be written, or more concise instructional material could be created. The theory of the subject could be studied further by using more varied sources.

Keywords: game-making in education, constructionism, games and learning, educational games

SISÄLLYS

LYHENTEET JA TERMIT

21st century skills	2000-luvulla tarvittavat taidot, joita kouluissa tulisi opettaa perinteisten lisäksi, esimerkiksi kriittinen ajattelu ja informaatiolukutaito.
Assetti	Englannin kielen sanasta *asset*, käytetään videopelien yhteydessä yläkäsittteenä kaikille pelin muodostaville rakennuspaloille. Pitää sisällään esimerkiksi tekstuurit, ääniefektit, musiikin, kooditiedostot ja kentät.
Behaviorismi	Teoria jonka mukaan tietoa voidaan siirtää opettajalta oppijalle. Oppiminen tapahtuu harjoittelemalla ja rohkaisemalla toivottua käytöstä.
COTS	*Commercial off-the-shelf*, viittaa kaupallisiin viihdepeleihin, joita ei ole suunniteltu opetuskäyttöön.
Konstruktionismi	Oppimisteoria jonka mukaan oppimista tapahtuu vuorovaikutuksessa muiden kanssa, rakentaessa jotain julkista ja sosiaalisesti merkittävää kokonaisuutta. Pohjautuu konstruktivismiin. (”N-sana”)
Konstruktivismi	Tiedonrakentumisteoria jonka mukaan yksilön käsitys ilmiöistä muodostuu ja rakentuu vuorovaikutuksessa ympäristön ja muiden ihmisten kanssa. (”V-sana”)
Pelieditori/-moottori	Käytetään joskus rinnakkaisina termeinä kuvaamaan pelintekoon tarkoitettua ohjelmaa. Tässä raportissa pelimoottori viittaa käyttäjälle näkymättömään teknologiaan, joka tarjoaa resurssit ja asettaa rajat luotavalla pelisisällölle. Editori viittaa moottorin näkyvään osuuteen, jonka avulla sisältöä luodaan.
Sandbox-peli	Peli joka antaa pelaajansa edetä niin kuin haluaa tiukan ennalta määrätyn kaavan sijaan.
Skene	Englannin sanasta *scene*, pelien yhteydessä viittaa niihin pelin muodostaviin ”kohtauksiin”, joiden välillä pelaaja liikkuu. Esimerkiksi kentät tai valikot voivat olla omat skenensä.

Skripti	Verrattain lyhyt koodikokonaisuus. Käytännössä teksti-tiedosto, joka pitää sisällään valitulla ohjelmointikielellä kirjoitetut komennot, jotka ohjelman tulisi suorittaa.
STEM	*Science*, *technology*, *engineering*, *mathematics*; luonnon-tieteiden, teknologian, insinööritaidon ja matematiikan alat.
TG	*TeacherGaming*, 5 More Minutesin brändi.
Web 2.0	Käsitys internetin nykytilasta, johon liittyvät olennaisesti sosiaalisuus ja oman sisällön tuottaminen ja jakaminen.

1 JOHDANTO

Pelialan tilastoihin keskittyvän SuperDatan raportin mukaan videopelit ja niihin liittyvä muu interaktiivinen viihde (mm. e-urheilu, videot) tuottivat vuonna 2017 108,4 miljardia dollaria (SuperData Research 2018). Statista-sivuston mukaan peliala tuotti samana vuonna maailmanlaajuisesti 104,57 miljardia dollaria. Tässä luvussa tosin otettiin huomioon vain laitteiden ja ohjelmistojen tuotot. Sivustolla esitetyn ennusteen mukaan ala tulee kasvamaan vuosittain, ennuste vuodelle 2021 on 138,4 miljardia dollaria. (Statista 2018.) WePC-sivuston kokoaman yhteenvedon perusteella pelaajia on maailmanlaajuisesti yli 2,5 miljardia (WePC 2018). Samankaltaisia lukuja pyöritellään myös muissa vastaavissa lähteissä: toisen katsauksen perusteella vuoden 2018 maailmanlaajuinen pelaaja-arvio on yli 2,3 miljardia ja ennuste vuodelle 2021 yli 2,7 miljardia (Filmora 2018).

Ei olekaan yllättävää, että myös kasvatustieteilijät ovat enenevissä määrin miettineet, kuinka tuoda videopelit tehokkaasti osaksi opetusta. Syy ei kuitenkaan ole pelkissä kasvavissa numeroissa. Pohjimmiltaan videopelit ovat järjestelmiä, joissa oppiminen on olennainen osa kokemusta. Edetessään pelaajat saavat jatkuvasti palautetta tekemisistään ja mukauttavat tämän perusteella pelityyliään. Videopelit ovat nykymaailmassa suosittua ajanvietettä ja motivaatio pelaamiselle tulee niistä itsestään, ilman ulkoisia pakotteita. Oikein käytettynä peleistä voidaankin saada tehokkaita opetuksen apuvälineitä.

Tämän työn kannalta opetuspelejä tärkeämmässä asemassa on kuitenkin pelinteko osana opetusta. Koska pelit ovat niin yleinen osa elämää, myös ajatus niiden tekemisestä innostaa monia. Pelinteko opettaa useita nykymaailmassa tärkeitä taitoja. Ilmeisimmät lienevät ohjelmointi- ja teknologian käyttöön liittyvät taidot. Lisäksi pelintekoon liittyy esimerkiksi sosiaalista vuorovaikutusta, ongelmanratkaisua, kriittistä ajattelua ja luovuutta. Se voi tarjota ensikosketuksen tietotekniikka-alaan ja kenties esitellä tulevaisuuden ammatin.

Opinnäytetyön toimeksiantaja on 5 More Minutes -opetuspeliyritys. Yrityksen tärkein tuote on TeacherGaming Desk -alusta, jonka kautta julkaistaan erilaisia opetuskäyttöön soveltuvia, useimmiten kolmannen osapuolen kehittämiä pelejä. Tätä kautta on julkaistu myös Data Realmsin kehittämän Planetoid Pioneers -pelin Contributor Edition -versio.

Se eroaa muista alustan peleistä siinä, että pääpaino on pelaamisen sijaan sisällön luomisella eli kyseessä on käytännössä pelieditori.

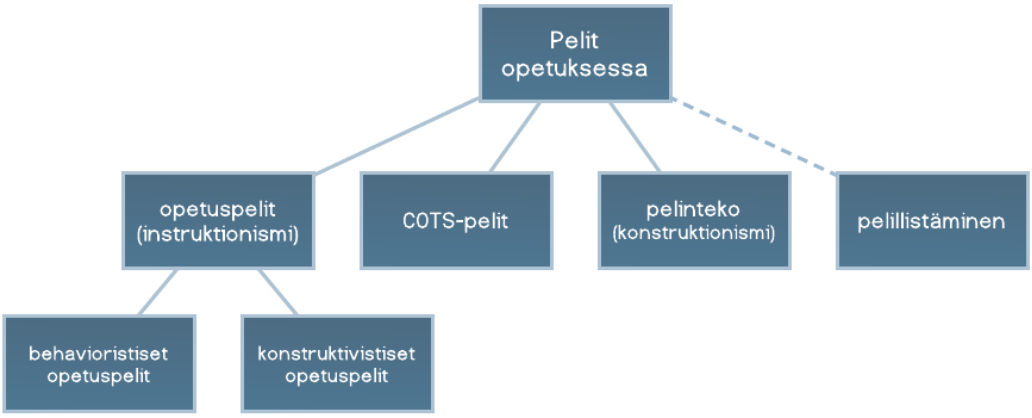
Planetoid Pioneersia ei ole suunniteltu opetuskäyttöön. Koska kyseessä on monimutkainen ohjelma, käyttöliittymä voi vaikuttaa sekavalta. Käyttäjät voivat luoda uutta sisältöä joko käyttämällä saatavilla olevia assetteja (engl. *asset*, viittaa niihin kaikkiin osasiin, jotka yhdessä muodostavat pelin) tai luomalla kokonaan uusia. Editorille ei ole saatavilla kattavaa aloitusopasta, vaan tutoriaaleja löytyy useilta eri verkkosivuilta sekä alkuperäisten pelinkehittäjien että itsenäisten sisällöntuottajien tekeminä. Näin ollen kynnys Planetoid Pioneersin käytön aloittamiselle opetustilanteessa on korkea. Opinnäytetyön tavoite oli tämän kynnyksen madaltaminen. Planetoid Pioneersista tehtiin opetuskäytössä ainakin hieman helpommin lähestyttävä. Tähän päästiin kokoamalla aloitusopas, jossa esitellään editorin ikkunoita, välilehtiä, käsitteitä ja pikanäppäimiä. Opas on tarkoitettu oppilaiden käyttöön ja kirjoitettiin englanniksi. Valmis opas on nähtävissä liitteessä 1.

Tämän raportin luvuissa 2 ja 3 käydään läpi pelien opetuskäytön teoriaa. Näistä ensimmäisessä käsitellään aihetta yleisemmällä tasolla, jälkimmäisessä tarkennetaan pelintekoon. Seuraavassa, luvussa 4, esitellään Planetoid Pioneersia ja arvioidaan sen vahvuuksia ja heikkouksia opetuskäytössä. Luvussa 5 kuvataan ohjeistuksen toteutustapaa ja valmista tuotosta. Työ päättyy luvun 6 pohdintaan.

2 VIDEOPELIT OPETUSKÄYTÖSSÄ

2.1 Käsitteitä ja lähestymistapoja

Pelejä voidaan käyttää opetuksessa usein eri tavoin. Ilmeisin lienevät opetuspelit, eli pelit, jotka on alusta alkaen suunniteltu opetuskäyttöön. On myös olemassa esimerkkejä, joissa kaupallisia pelejä (kirjallisuudessa usein *COTS, commercial off-the-shelf*) on onnistuneesti tuotu luokkahuoneisiin. Kolmas lähestymistapa on (opetuksellisten) pelien tekeminen oppilaiden toimesta. Pelien opetuskäytöstä puhuttaessa esille nousee usein myös pelillistäminen, eli pelielementtien tuominen tosielämään. Tähän ei kuitenkaan suoraan liity pelejä. Kuviossa 1 on nähtävissä nämä lähestymistavat, seuraavilla sivuilla kutakin käsitellään hieman tarkemmin. Tämän työn osalta olennaisinta on pelinteko, johon perehdytään sille omistetussa luvussa.



KUVIO 1. Lähestymistapoja pelien käyttöön opetuksessa

Usein puhutaan niin kutsutusta piilo-oppimisesta (engl. *stealth learning*), jolla viitataan siihen, että pelaajat oppivat uutta huomaamattaan. Jotkut opettajat ja pelisuunnittelijat jopa uskovat, että oppilaat välttävät pelejä, joiden tietävät olevan opettavaisia (Turkay & Adinolf 2012, 3348). Sekä Whitton että Turkay ja Adinolf kuitenkin suhtautuvat tähän termiin varauksella. ”Salaa” opittu tieto ei Whittonin (2014) mukaan ole pysyvää ja sitä on hankalaa soveltaa muissa konteksteissa. Toisaalta tahatonta oppimista tapahtuu koko ajan. (Whitton 2014, 43, 166.) Turkayn ja Adinolfin tutkimus taas viittaa siihen, että pelaajia ei häiritse peleistä oppiminen vaan ennemminkin, kuinka opetuksellisuus on toteutettu. Jos peli on suunniteltu hyvin, sitä ollaan valmiita pelaamaan sisällöstä riippumatta. (Turkay & Adinolf 2012, 3348.)

Kuviossa 1 opetuspelit jaetaan edelleen behavioristisiin ja konstruktivistisiin opetuspeleihin. Nämä ovat Whittonin (2014) esittelemät yleisimmät opetuspelityypit. Behavioristisen käsityksen mukaan mielen voi täyttää tiedolla siirtämällä sitä opettajalta oppijalle, harjoittelulla ja vahvistamalla toivottua käytöstä. Konstruktivismi sitä vastoin lähtee yksilön, tässä tapauksessa oppijan, näkökulmasta. Sen mukaan jokainen rakentaa ilmiöstä (opittavasta aiheesta) omanlaisensa käsityksen, joka kehittyy vuorovaikutuksessa ympäristön ja muiden ihmisten kanssa. (Whitton 2014, 27–28.) Opetuspelit ovat omanlaistaan ohjaavaa opetusmateriaalia. Näin ollen pelaaminen voidaan nähdä instruktionistisena tapana tuoda pelit opetukseen (Kafai 2006, 36–37). Tämä luokittelu on tosin hieman yksinkertaistettu, sillä Kafai tuntuu tekstissään viittaavan lähinnä behavioristisiin opetuspeleihin. Tästä eteenpäin tässä työssä kuitenkin käytetään tätä yleistystä.

Behavioristisissa peleissä oppiminen pohjautuu faktojen opetteluun ja muistamiseen käytännössä soveltamisen sijaan. Oppimista ei ole yhdistetty saumattomasti pelaamiseen, vaan oppisisällön päälle on ennemminkin lisätty uusia, ulkoisesti motivoivia elementtejä (ns. ”suklaakuorrutettu parsakaali”). Ne ovat yleisiä, koska niiden suunnittelu ja toteutus on helppoa ja halpaa ja vaikutus oppimiseen on helposti arvioitavissa. Konstruktivistisissa peleissä oppisisältö on sulautettu hienovaraisemmin osaksi pelikokemusta. Pelaajan kohtaamat haasteet ja tehtävät vastaavat opetuksen tavoitteita, mutta niillä on pelimaailmassa konteksti ja motivaatio niiden suorittamiseen tulee pelin sisältä. Vaikka on olemassa joitain lupaavia tutkimuksia tällaisten pelien toimivuudesta, ne ovat behavioristisia harvinaisempia, sillä niiden opettamat taidot kehittyvät hitaammin ja ovat vaikeammin mitattavissa. (Whitton 2014, 25–26, 28–29.)

Kouluissa voidaan hyödyntää myös COTS-pelejä, joita ei ole suunniteltu opetuskäyttöön. Tällaiset pelit ovat usein opetuspelejä motivoivampia, mutta niiden yhteensopivuudesta opetussuunnitelman kanssa ei ole takeita. Lisäksi ne ovat usein kalliita ja vaativat toimiakseen tehokasta laitteistoa. (Whitton 2014, 26.) Ongelmista huolimatta joitain onnistuneitakin esimerkkejä on, kuten SimCity 2000 -kaupunginrakennuspeli (tutkimus vuodelta 1998), historiallinen strategiapeli Civilization III (2004) sekä Neverwinter Nights -roolipeli (2008) (Turkay & Adinolf 2012, 3346; Whitton 2014, 26). Opetuskäyttöön siirtämisen helpottamiseksi COTS-peleistä voidaan myös tehdä erillisiä opetusversioita, TG Desk -alustalla on saatavilla tällaiset versiot esimerkiksi Kerbal Space Program -avaruuslentosimulaattorista ja Cities Skylines -kaupunginrakennuspelistä.

Vaikka edellä esiteltiinkin kolmea eri pelityyppiä, käytännössä tutkimus on keskittynyt näistä lähinnä ensimmäiseen ja viimeiseen. Kafain ja Burken (2015) mukaan opetuspelitutkimuksen suuri haaste on pitkään ollut oikean tasapainon löytäminen toistoon ja ulkoa opetteluun pohjautuvien ja COTS-pelien välillä. He itse ovat sitä mieltä, että pelien opetuspotentiaali löytyy ennemminkin jostain pelaamisen ja tekemisen väliltä, oppilaiden itse tekemistä opetuspeleistä. (Kafai & Burke 2015, 313–314.)

Tämä on konstruktionistinen tapa lähestyä aihetta. Nimi muistuttaa syystäkin aiemmin esiteltyä konstruktivistista tiedonrakentumisteoriaa, oppimisteorian kehittäjä nimittäin työskenteli erään liikkeen johtohahmon kanssa ja pohjasi ajatuksensa tähän. Myös konstruktionismin mukaan oppiminen tapahtuu käsityksiä rakentamalla ja muokkaamalla vuorovaikutuksessa muiden kanssa. Tämän lisäksi olennaisessa asemassa on jonkin sosiaalisesti merkittävän, julkisen kokonaisuuden (esimerkiksi pelin) luominen. (Kafai 2009, 35–36.) Whitton nimesi pelinteon yhdeksi niistä opetuksen suuntauksista, jonka uskoi kasvavan lähitulevaisuudessa (Whitton 2014, 189).

Pelillistämisellä tarkoitetaan pelillisten elementtien, kuten pisteytyksen ja tulostaulukoiden, tuomista erilaisiin tilanteisiin. Sen toteuttaminen on verrattain helppoa ja halpaa, mutta se ei muuta itse opetuksen sisältöä, vaan lisää sen päälle ulkoisen motivaatiotekijän. (Whitton 2014, 85.) Edut ja ongelmat ovat siis samankaltaisia kuin behavioristisissa opetuspeleissä. Aiempaa suklaaparsakaalimetaforaa muistuttaen muuan opetuspeleihin ja -simulaatioihin keskittyvän yrityksen toimitusjohtaja A. Koop kuvaili Nordic Game -konferenssissa pitämällään luennolla pelillistämistä ”kuorrutteeksi surkean kakun päällä” (Koop 2018).

Pelialan mittavuus ja videopelien teoreettiset hyödyt heijastuvat monina tapoina, joilla pelejä on pyritty valjastamaan opetuskäyttöön. Ei ole selkeää vastausta, mikä niistä on tehokkain. Behavioristiset opetuspelit ja pelillistäminen lienevät helpoiten toteutettavissa, mutta samalla vähiten motivoivia ja näin osin kumoavat yhden videopelien useimmin noteeratuista hyödyistä. Pelit joita ei ole suunniteltu opetuskäyttöön voivat puolestaan innostaa oppilaita, mutta eivät välttämättä vastaa opetussuunnitelmia. Jotkut itse asiassa uskovat, että pelkkä tieto opettavaisuudesta vaikuttaa negatiivisesti haluun pelata. Tämä olettamus saattaa kuitenkin olla virheellinen, karsastaminen voi johtua ennemminkin opetussisällön toteutustavasta kuin olemassaolosta. Toisaalta mikäli pelaaja ei tiedosta opetussisältöä, ei se välttämättä jää mieleen. Ainakin teoriassa hieman kau-

pallisia muistuttavat, konstruktivistiset opetuspelit voisivatkin olla paras tapa tuoda videopelit luokkahuoneisiin. Käytännössä tämä on kuitenkin työläisyytensä takia harvinaista. Jotkut tutkijat ovat ehdottaneet, että vastaus voisikin piillä oppilaiden itse tekemissä peleissä.

2.2 Miksi käyttää videopelejä opetuksessa?

Viihdyttävyys on tärkeä osa hyvin suunniteltua peliä, oli se sitten kaupallinen tai opetuksellinen. Hauskuus voi kuitenkin olla olennainen osa myös oppimiskokemusta. Whitton lainaa kirjassaan julkaisuja, joista toinen käsittelee aivokemiaa ja toinen neurologiaa ja joista molemmissa tunnistettiin yhteys uuden oppimisen ja iloisuuden tunteiden välillä (Whitton 2014, 114–115). Toisaalta oppiminen on aina osa pelikokemusta, sillä haasteet ovat osa peliä. Usein pelikirjallisuudessa (mm. Whitton 2014, 96) puhutaan haasteiden ja taitojen tasapainon tärkeydestä: mikäli peli on liian helppo, pelaaja tylsistyy. Jos se taas on liian vaikea, turhauttaa se pelaajaansa. Koska pelaaja oppii koko ajan, hyvin tasapainotetussa pelissä haasteet vaikeutuvat pelaajan kykyjen kasvaessa.

Kuten missä tahansa ohjelmistossa, myös peleissä tilanteeseen sopiva, oikein ajoitettu palaute on tärkeä osa hyvää kokemusta. Whittonin (2014) mukaan tämä on yksi niistä ominaisuuksista, jotka tekevät niistä tehokkaita työkaluja opetuksessa. Ne tukevat jatkuvaa oppimista antamalla automaattista, räätälöityä palautetta. Aina kun pelaaja tekee jonkin toiminnon, oli se sitten pieni tai suuri, peli reagoi jotenkin. Tämän reaktion, onnistumisen tai epäonnistumisen, perusteella pelaaja arvioi tilannettaan ja päättää seuraavan liikkeensä. Siinä missä virallisessa opetuksessa virheet nähdään usein epäonnistumisina, peleissä ne ovat osa (oppimis-) kokemusta, mikä sekin on yksi niiden eduista. (Whitton 2014, 39, 133, 148.)

Koska hauskuudella on paikkansa oppimisessa ja taitojen kehityksellä omansa peleissä, on peleillä selvästi potentiaalia opetuskäytössä. Niiden käytön tutkimus- ja teoriakenttä on kuitenkin todellisuudessa sirpaleinen ja kiistattomat todisteet niiden hyödyistä puutteelliset (Whitton 2014, 21; de Freitas 2018, 74). Useat tutkimukset kuitenkin viittaavat siihen, että pelit lisäävät oppilaiden motivaatiota ja uppoutumista. Monet pelaavat vapaaehtoisesti pelaamisen itsensä takia, ilman ulkoisia kannustumia. Motivaatio kumpuaa pelien sisältä. (Whitton 2014, 21, 69.) Ainakin teoriassa tämä sama into voitaisiin tuoda opetuspeleillä osaksi opetusta.

Pelit luovat ympärilleen maailmoja, joihin uppoutua. Näissä maailmoissa annetuilla haasteilla on merkitys ja konteksti, eivätkä ne ole vain irrallisia, abstrakteja tehtäviä. Tämä on yksi pelien eduista verrattuna perinteiseen opetukseen, varsinkin jos annetut haasteet vastaavat opetuksen tavoitteita. (Whitton 2014, 41.) Samankaltaisia etuja on huomattu myös pelien teossa. Vuonna 1995 tehdyssä tutkimuksessa havaittiin, että pelejä tehneet oppilaat oppivat enemmän ohjelmointikonsepteista kuin pienempiä projekteja tehneet (Kafai & Burke 2015, 318). Pelintekoon liittyviin hyötyihin perehdytään tarkemmin alaluvussa 3.2.

Videopeleille on siis luontaista opettaa pelaajaansa, hyvin suunnitellussa pelissä kehityksen jatkuva tukeminen on keskeinen osa kokemusta. Oppimista tuetaan saumattomalla palautteella ja virheiden teko on sallittua. Pelit tarjoavat haasteita, joilla on kontekstissaan merkitys ja motivaatio niiden suorittamiselle tulee peleistä itsestään. Jos nämä haasteet vielä vastaavat opetuksen tavoitteita, voivat pelit ainakin teoriassa olla merkittävä oppimisen edistäjä. Tätä teoriaa ei kuitenkaan ole aukottomasti onnistuttu osoittamaan todeksi, vaikka lupaavia tutkimuksia onkin olemassa. Useat tutkijat ovat kuitenkin yhtä mieltä siitä, että videopelit lisäävät oppilaiden motivaatiota.

2.3 Haasteet

Kuten edellä mainittiin, yksi pelien opetuskäytön ongelmista on kirjallisuuden epäjohdonmukaisuus. Kiistaton, opetuspelien tehokkuuden todistava tutkimuspohja puuttuu. Joidenkin tutkimusten mukaan peleillä on positiivinen vaikutus, toisissa ei havaittu eroja. Tulosten ristiriitaisuutta selittävät monet seikat: käytetyissä tutkimusmetodeissa on eroja ja puutteita, käsitteitä ei käytetä johdonmukaisesti, eivätkä eri tutkimustilanteet ole suoraan verrattavissa keskenään. (Whitton 2014, 20–22.) De Freitas tunnisti 2018 julkaistussa artikkelissaan samanlaisia ongelmia: ei ole yhtä selkeää peleihin ja opetukseen keskittyvää tieteenalaa, vaan aihetta on tutkittu vuosien varrella useista eri näkökulmista. Hän kuitenkin tuli siihen tulokseen, että puutteista huolimatta tutkimukset osoittavat peleistä saatavat hyödyt. Lisäksi hän arvioi alan teorian ja tutkimuksen selkeytyneen ja vakautuneen viime vuosina. (de Freitas 2018, 74, 76, 80.)

Vaikka peleillä onkin havaittu olevan positiivinen vaikutus motivaatioon, olisi liioiteltua julistaa ne ehdottomaksi, kaikille sopivaksi oppimisen edistäjäksi. Opetustilanteessa pelaaminen lakkaa olemasta vapaaehtoista ja motivaatiosta tulee ainakin osittain ulkois-

ta. Toisaalta kaikki eivät innostu joistain pelityypeistä tai videopeleistä yleensä. Uppoutuminen ei myöskään aina tarkoita oppimista, vääriin asioihin keskittyminen voi itse asiassa haitata sitä. (Whitton 2014, 69–70.) Viimeksi mainittuja ongelmia on havaittu sekä instruktionistisessa että konstruktionistisessa lähestymistavassa. Pelejä tehdessä oppilaat saattavat keskittyä enemmän pelimaailmojensa luomiseen kuin suunniteltuun opetussisältöön (Kafai & Burke 2015, 321 (lähteenään F. Ke 2014)).

Monilla saattaa olla ennakkokäsityksiä peleistä. Esimerkiksi pelien ja väkivaltaisen tai muuten aggressiivisen käytöksen välistä suhdetta on puitu mediassa jo vuosia. Tutkimuksissa tätä yhteyttä ei ole onnistuttu todistamaan oikeaksi tai vääräksi, jotkut puhuvat sen puolesta, toiset sitä vastaan. Lisäksi pelit, niin harrastuksena kuin alana, nähdään usein ennemmin pojille ja miehille kuin tytöille ja naisille suunnattuna. Kolmas ongelmallinen aihe on peliriippuvuus. Jotkut näkevät pelaamisen voivan aiheuttaa addiktiota, mutta tästäkään ei ole vankkoja todisteita. Toisinaan ongelmat voivat piillä oppijoiden itsensä asenteissa, varsinkin aikuisopiskelijat voivat nähdä videopelit ajanhukkana. Asenteiden, sekä vanhempien ja opettajien että joissain tapauksissa oppilaidenkin, muuttaminen tulee siis olemaan tärkeä osa pelien opetuskäytön yleistymistä. (Whitton 2014, 186–189.)

Selkeistä teoreettisista hyödyistä huolimatta pelien opetuskäyttö kohtaa käytännössä monia haasteita. Vaikka merkkejä kehityksestä parempaan suuntaan on, alan kirjallisuus on hajanaista, johdonmukaisimmin todisteita on saatu vaikutuksesta motivaatioon. Motivaatio ei kuitenkaan takaa oppimista taikka toisaalta ole itsestäänselvyys. Joitain pelaaminen ei kiinnosta lainkaan, toisten innokkuutta haittaa pakollisuus. Myös asenteet videopelejä kohtaan vaihtelevat suuresti, alan innokkaimmille kannattajille ne ovat alihyödynnetty voimavara, jotkut pitävät niitä joutavana ajanhukkana, toiset saattavat nähdä ne haitallisina, miltei vaarallisina. Ongelmista huolimatta videopelien puolestapuhujat uskovat, että niillä tulee olemaan tärkeä rooli tulevaisuuden opetuksessa.

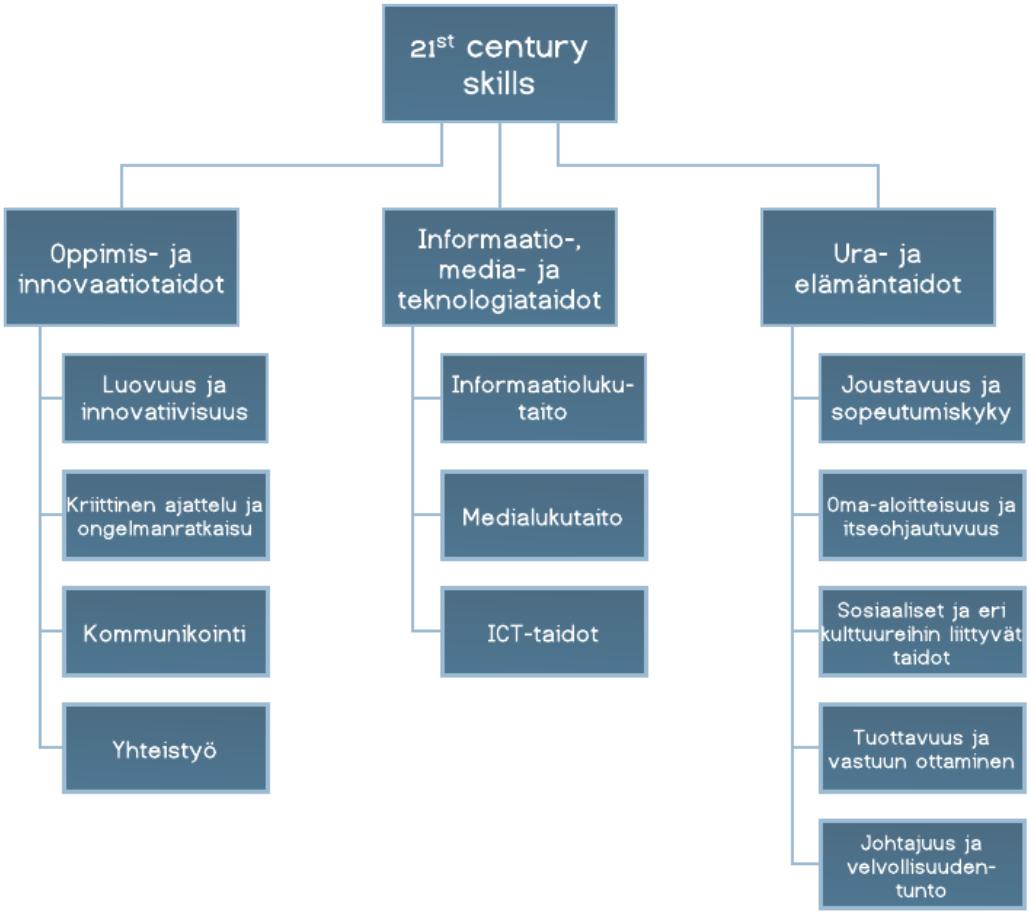
2.4 Opetuksen tulevaisuus ja ”tietokoneajattelu”

Whitton kuvailee kirjansa päätösluvussa tulevaisuuden ideaalia maailmaa, jossa koulutuksen päätavoite on itsenäisten ajattelijoiden ja oppijoiden luominen. Tähän päästään panostamalla nykymaailmassa tärkeiden taitojen (kuten digitaalinen ja informaatiolukutaito) opetteluun tarkkaan määrättyjen sisältöjen sijaan. Tässä maailmassa oppijat ratko-

vat yhdessä ongelmia, joilla on heille itselleen merkitys ja virheiden tekeminen on osa oppimiskokemusta. Niin oppilaat kuin opettajatkin etsivät aktiivisesti uusia mahdollisuuksia ja ratkaisuja sekä luovat uutta. Teknologia on olennainen osa opetusta, joten kyky käyttää sitä on välttämätön. Oli tulevaisuus mikä hyvänsä, todellisuutta on, että ihmisillä tulee olla kyky reagoida odottamattomaan ja oppia uutta alati muuttuvassa maailmassa, jossa työpaikat vaihtuvat ja toisinaan katoavat kokonaan. (Whitton 2014, 182–185.)

Koop päätti luentonsa samankaltaisiin sanoihin. Myös hän kuvailee tulevaisuutta, jossa työpaikkoja automatisoidaan ja ihmisiä korvataan tekoälyllä ja roboteilla. ”Tietämisen” arvo laskee, kun tarvitsemansa tiedon voi hakea internetistä. Tällaisessa maailmassa opetuksen ei tulisi keskittyä faktojen muistamiseen, vaan ihmisten tulisi kyetä arvioimaan informaatiota kriittisesti. Heidän pitää pystyä ongelmanratkaisuun sekä luovaan ja innovatiiviseen ajatteluun. Koop uskoo, että peleillä on tärkeä asema tällaisen maailman opetuksessa. Kokemuksiin perustuva oppiminen on tehokasta, koska uusissa tilanteissa aivot rakentavat jatkuvasti yhteyksiä asioiden välille. Ne eivät kuitenkaan erota todellista ja simuloitua, pelissä tapahtuvaa kokemusta. Peleissä oppilaat voivat kokeilla, ottaa riskejä ja tehdä virheitä hallitussa ympäristössä. ”Games have the power to rewire our brains, and that’s why the future of education is games-based.” (”Peleillä on kyky saada aivomme rakentamaan uudenlaisia yhteyksiä, tämän takia opetuksen tulevaisuus on peleissä.”) (Koop 2018.)

Myös de Freitasin artikkelissa sivutaan tulevaisuuden oppimista. Hän viittaa vuoden 2014 julkaisuunsa, jossa listasi perinteisiä, uusia ja tulevaisuuden oppimisen lähestymistapoja. Myös hän näkee oppimisen siirtyvän luokkahuoneiden ulkopuolelle, saumattomaksi, jatkuvaksi osaksi elämää. Uudessa lähestymistavassa opetuksen sisällöt kehittyvät perinteisistä opetussuunnitelmista kohti 2000-luvulla tarvittavien taitojen (engl. *21st century skills*) opetusta, tulevaisuudessa kenties yksilöllisempään opetukseen. Opetus muuttuu oppilaslähtöisemmäksi ja pohjautuu haasteisiin ja ongelmanratkaisuun. (de Freitas 2018, 77.) Kuvio 2 esittää eräässä tunnetussa viitekehyksessä listattuja taitoja, joita tulisi opettaa perinteisempien lisäksi.



KUVIO 2. 2000-luvulla tarvittavia taitoja (Partnership for 21st Century Learning 2015)

Näiden tulevaisuudenkuvien perusteella opetuksen tavoite tulee siirtymään pois pelkkien faktojen opettelusta, kohti 2000-luvulla tärkeiden taitojen, kuten kriittisen ajattelun, ICT-, innovaatio-, ongelmanratkaisu- ja informaatiolukutaitojen, kehitystä. Opetus muuttuu oppilaslähtöisemmäksi ja pitää sisällään oppilaille itselleen merkittäviä haasteita. Jatkuvasta oppimisesta tulee osa maailmaa, jossa työnkuvat vaihtuvat ja katoavat ja tietotyö korvaa fyysisen työn. Koulujen tulisi valmistaa oppilaansa tähän. Peleillä on paikkansa osana tällaisen maailman opetusta.

”Tietokoneajattelu” (engl. *computational thinking, CT*) on J. Wingin artikkelissaan vuonna 2006 esittelemä ajatus nykymaailmassa tärkeästä taidosta. Artikkelilla oli merkittävä vaikutus ICT-taitojen opetuksen tärkeyttä korostavaan liikkeeseen, vaikka se ei määritellytkään, miten tällainen ajattelu käytännössä tuotaisiin opetukseen tai mitä se tarkalleen edes tarkoittaa (Howland & Good 2015, 225; de Paula, Burn, Noss & Valente 2018, 39). Wing kuvaa tietokoneajattelua ”tietojenkäsittelytieteelliseksi tavaksi lähestyä ongelmanratkaisua, suunnitella järjestelmiä sekä ymmärtää ihmisten käytöstä”. Se ei siis

ole synonyymi ohjelmointitaidoille. Hän rinnastaa sen tärkeydessään luku-, kirjoitus- ja aritmeettisiin taitoihin yhteiskunnassa, jossa teknologia on läsnä kaikkialla. Nimestään huolimatta tietokoneajattelu ei tarkoita, että ihmisten tulisi pyrkiä ajattelemaan kuin koneet. Sen sijaan se pitää sisällään uusien, mielikuvituksellisten ratkaisujen kehittämiseen teknologiaa hyödyntäen. (Wing 2006, 33–35.)

Vaikka tämä ei olekaan yhtä laaja kuvaus tulevaisuudesta kuin aiemmat tai lähesty aihetta videopelien kautta, ajatuksilla on yhtymäkohtia. Tällaisen ajattelun voisi itse asiassa katsoa kattavan useampia, edellä esiteltyjen mainitsemia tulevaisuudessa tarvittavia taitoja. Ilmeisin näistä lienevät teknologiankäyttötaidot. Lisäksi se pitää sisällään ongelmanratkaisutaitoja ja luovuutta. Tietokoneajattelussa siis yhdistyy useita 2000-luvun taitoja. Pelinteko on yksi keino, jolla tällaista ajattelua on lähdetty esittelemään oppilaille.

3 PELINTEKO OPETUKSEN OSANA

3.1 Konstruktionismi ja pelinteon ajankohtaisuus

Konstruktionismi asetetaan usein vastakkain perinteisemmän, opettajan johdolla tapahtuvan instruktionistisen opetustavan kanssa. Oletetaan, että oppilaslähtöinen oppimisteoria tarkoittaa muodollisen opetuksen hylkäämistä kokonaan ja siirtymistä itsenäiseen opiskeluun. (Kafai 2009, 36.) Tämä on kuitenkin liioittelua, sillä vuorovaikutus opettajan ja oppilaiden välillä on osa myös tätä lähestymistapaa, opettajan rooli tosin muuttuu tiedon jakajasta oppimisen fasilitaattoriksi (Kafai 2009, 36; Whitton 2014, 127).

Kafain karkean jaottelun mukaan opetuspelit edustavat instruktionismia, peliteko konstruktionismia. Vaikka onkin tiedostanut, etteivät näkemykset ole toistensa vastakohtia ja että molemmilla on paikkansa, hän on konstruktionismin puolestapuhujana kritisoinut joitain opetuspelien ominaisuuksia. Hän esimerkiksi kyseenalaistaa, millaisen kuvan oppijat saavat oppimisesta, kun vaikeista aiheista pyritään tekemään helppoja ja hauskoja. (Kafai 2006, 36–38; Kafai 2009, 36.) Huomautettakoon tosin, että tuoreemmassa artikkelissa, jota hän oli mukana kirjoittamassa, pelintekoa puolusteltiin samankaltaiselta kritiikiltä. Jotkut kokevat, että yksinkertaistettu tapa, jolla ohjelmointia esitellään oppilaille, antaa siitä väärän kuvan (Kafai & Burke 2015, 318). Lisäksi Kafai toi vanhemmassa tekstissään esille opetuspelien tutkimuksen puutteet, mutta vuonna 2015 julkaistun artikkelin perusteella myöskään pelinteko ei ole välttynyt tältä ongelmalta (Kafai 2006, 27; Kafai & Burke 2015, 325–326).

Tämän vertailun perusteella ei siis voi väittää, että toinen lähestymistavoista olisi parempi. Pelinteko kuitenkin opettaa erilaisia asioita kuin pelaaminen ja on tällä hetkellä ajankohtainen tapa lähestyä aihetta. Whitton (2014) uskoi, että luovuus tulee olemaan kasvava osa pelien opetuskäyttöä. Omalta osaltaan tätä edesauttavat monet tavat, joilla nykypelit mahdollistavat oman sisällön luomisen. Useat pelit tarjoavat työkalut esimerkiksi omien kenttien tekemiseen. Myös modaaminen, eli muutosten tai lisäysten tekeminen olemassa oleviin peleihin (tai toisinaan täysin uuden pelin tekeminen vanhan päälle) on yleistä. Lisäksi esimerkiksi Minecraftin kaltainen suosittu sandbox-peli antaa pelaajalleen vapaat kädet tehdä mitä haluaa. (Whitton 2014, 127, 129.) Minecraft onkin oiva esimerkki, kuinka menestyviä luovuuteen ja itse tekemiseen perustuvat pelit voivat

olla: sillä oli tämän vuoden alussa julkaistun artikkelin mukaan yli 74 miljoonaa kuukausittaista käyttäjää ja sitä oli myyty miltei 150 miljoonaa kappaletta (Gilbert 2018).

Samankaltaisia näkemyksiä on myös Kafain ja Burken teksteissä. Siinä missä peliala ennen suhtautui nihkeästi kuluttajien tekemään sisältöön, nykyään se on osa useita menestyviä pelejä (Kafai & Burke 2015, 314). Tämä innoittaa tuomaan itse tekemisen myös luokkahuoneisiin. Toisaalta oman sisällön luominen (ja myös konstruktionismille olennainen sosiaalinen vuorovaikutus) on olennainen osa tämänhetkistä internetkulttuuria (ns. Web 2.0) ja sitä kautta jokapäiväistä elämää (Burke & Kafai 2013).

Itse tekeminen ja osallistuminen ovat merkittäviä osia nykypäivän mediakulttuuria. Tämä näkyy niissä monissa tavoissa, joilla videopelit kannustavat pelaajiaan luomaan omaa sisältöään. Vaikka ei voidakaan väittää, että konstruktionismi olisi instruktionismia parempi tapa tuoda pelit opetukseen, lähestymistapa on kiistatta ajankohtainen. Planetoid Pioneersin kaltainen pelieditori voisi siis olla tärkeä osa 5 More Minutesin tarjontaa.

3.2 Pelinteon hyödyt

Pelinteko voi kehittää useita alaluvussa 2.4 esiteltyjä 2000-luvulla tärkeitä taitoja. Tietoteknisten taitojen lisäksi se vaatii luovuutta, kriittistä ajattelua, ongelmanratkaisua, kommunikointia, yhteistyötä ja itseohjautuvuutta. Pelinteko voi opettaa myös perinteisempiä sisältöjä. Koska opetustilanteessa tavoitteena on usein opetuspelin tekeminen (esim. Baytak & Land 2010) tai sisältö on muuten ennalta määrätty (esim. de Paula ym. 2018), tekijät oppivat pelin käsittelemästä aiheesta. Alaluvussa myös ennustettiin opetuksen muuttuvan oppilaslähtöisemmäksi. Pelinteko voi osaltaan edesauttaa tätä kehitystä antamalla oppilaille mahdollisuuden luoda omia, henkilökohtaisia projekteja (Howland & Good 2015, 227).

ICT-taidot kehittyvät, sillä pelinteko tutustuttaa oppilaat ohjelmoinnin perusteisiin, joko oikean tekstipohjaisen tai yksinkertaisemman, visuaalisen ohjelmointikielen kautta. Jo yhdessä aikaisimmista aiheen tutkimuksista (vuodelta 1995) huomattiin, että pelejä tehneet oppilaat oppivat monipuolisemmin ohjelmoinnista kuin pienempiä projekteja tehneet (Kafai & Burke 2015, 318). Pelintekoon liittyy taiteellisuutta ja suunnittelutehtäviä (Howland & Good 2015, 227). Se siis kannustaa luovuuteen ja innovatiivisuuteen. Op-

pilaat joutuvat arvioimaan omia ajattelutapojaan ja ratkaisemaan ongelmia; he suunnittelevat, tekevät päätöksiä ja etsivät ja korjaavat virheitä, koodin lisäksi omassa ajattelussaan (Kafai & Burke 2015, 321). Näin pelinteko kehittää itsenäisen, kriittisen ajattelun ja ongelmanratkaisun taitoja. Videopelit ovat yleensä useamman ihmisen yhteistyön tuloksia, joten niiden tekemiseen liittyy myös sosiaalista vuorovaikutusta, kommunikointia ja yhteistyötä. Vaikka pelinkehitys tapahtuisikin opetustilanteessa yksin, vuorovaikutus muiden kanssa on silti osa prosessia. Oppilaat esimerkiksi kysyvät toisiltaan neuvoja, testaavat toistensa pelejä ja saavat niistä vaikutteita (Baytak & Land 2010, 5244–5245).

Kafai ja Burke vetivät vuonna 2015 julkaistussa artikkelissaan yhteen 55 aiheesta vuosina 2005–2015 julkaistua tutkimusta. Edellä lueteltujen taitojen kehittymisen lisäksi näistä useissa havaittiin, että oppilaiden mielenkiinto alaa kohtaan kasvoi. Tähän oli yksi poikkeus, erään tutkimuksen perusteella pelinteko ei vaikuttanut tyttöjen haluun työskennellä matemaattis-luonnontieteellisillä STEM-aloilla. Kirjoittajat arvioivat tämän voivan johtua siitä, että alan nähdään suhtautuvan naisiin negatiivisesti, mutta tarkemmin mahdollisia syitä ei pohdita. (Kafai & Burke 2015, 319–320, 326.)

Aiemmassa tulevaisuutta käsittelevässä luvussa kuvailtu tietokoneajattelu tuodaan usein esille konstruktionistisissa julkaisuissa (mm. Howland & Good 2015; Kafai & Burke 2015; de Paula ym. 2018). Vaikka se ei tarkoitakaan pelkkää ohjelmointia, esimerkiksi Kafai ja Burke uskovat, että ohjelmointi on hyvä tapa sen esittelylle. He kuvaavat pelisuunnittelua ”optimaaliseksi varhaiseksi tietokoneajattelun hautomoksi”. (Kafai & Burke 2015, 316.) Samaiset tutkijat ovat myös esitelleet oman rinnakkaisen käsitteensä, ”computational participation”, joka kattaa lisäksi teknologian ympärille syntyvät yhteisöt ja niihin osallistumisen (Kafai 2016, 26–27). Oli käsite kumpi hyvänsä, se pitää sisällään useita nyky-yhteiskunnassa tärkeitä taitoja.

Pelien tekeminen tarjoaa usein mielekkään tavan tutustua aiheeseen. Videopelejä pelaavat ”kaikki”, taustaan tai ikään katsomatta (Kafai & Burke 2015, 325). Kiinnostus videopelejä kohtaan on yksi tekijä, joka saattaa innostaa tekemään niitä myös itse, mutta ei suinkaan ainut; motivaattoreina saattavat toimia myös pelintekoon liittyvä taiteellisuus, suunnittelutehtävät ja tarinankerronta (Howland & Good 2015, 226–227).

Koska pelit ovat nykymaailmassa niin yleinen harrastus, myös lupaus niiden tekemisestä voi innostaa. Tämä ei kuitenkaan ole ainut motivoiva tekijä. Pelit ovat monimutkaisia

järjestelmiä, joiden tekemiseen tarvitaan monenlaista osaamista. Kehitysprosessiin osallistuminen voi olla mielekästä myös niille oppilaille, joita videopelit itsessään eivät kiinnosta. Toisaalta pelinteko tarjoaa mahdollisuuden tehdä henkilökohtaisesti merkitseviä projekteja. Itse tekemisen kautta oppilaille voidaan esitellä useita muuttuvassa yhteiskunnassa tärkeitä taitoja. Ilmeisimpien ohjelmointi- ja ICT-taitojen lisäksi se opettaa esimerkiksi ongelmanratkaisua, kriittistä ajattelua, sosiaalisia taitoja ja kommunikointia. Mikäli pelin aihe on ennalta määrätty, voi tekeminen opettaa myös perinteisemmistä asiasisällöistä. Käytännössä pelintekoa on esitelty oppilaille erilaisten yksinkertaistettujen pelitekoympäristöjen avulla.

3.3 Hyvän opetuksellisen pelintekoympäristön ominaisuuksia

Hyvä käytettävyys on tärkeä osa mitä tahansa ohjelmistoa, oli kyseessä sitten hyötysovellus, videopeli tai, kuten tässä tapauksessa, pelieditori. ISO-standardijärjestö määrittelee käytettävyyden käsitteen seuraavasti: “Extent to which a system, product or service can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use” (ISO 9241-11:2018 2018, 3.1.1).

Selkeämmin sanottuna, tietokoneohjelman tapauksessa, se viittaa siihen, kuinka helppokäyttöinen ja käyttäjälleen hyödyllinen jokin sovellus on. Määritelmässä korostuu myös käyttötilanteen ja käyttäjien vaikutus käytettävyyden kokemiseen. Tässä tapauksessa käyttötilanne on opetustilanne ja käyttäjät opettajia ja oppilaita. Erään tunnetun käytettävyysasiantuntijan, Jakob Nielsenin, mukaan käytettävyyden kokemiseen vaikuttaa viisi osatekijää: opittavuus, tehokkuus, muistettavuus, virheet sekä tyytyväisyys (Nielsen 2012). Koska Planetoid Pioneersia ei ole alkujaan suunniteltu opetuskäyttöön, se ei käytettävyytensä puolesta ole optimaalinen tähän tarkoitukseen. Aiheeseen palataan tarkemmin alaluvussa 4.2.

Burke ja Kafai tuovat (M. Resnickiä ja tämän MIT-tiimiä mukaillen) esille kolme periaatetta, joita opetuskäyttöön suunniteltujen pelinteko- (ja muiden luovien) ympäristöjen tulisi noudattaa. Niissä tulisi olla ”matalat lattiat”, ”korkeat katot” ja ”leveät seinät”. Ensimmäinen viittaa siihen, että käytön aloittamisen tulisi olla helppoa käyttäjän aiemmasta kokemuksesta riippumatta. Korkeilla katoilla tarkoitetaan sitä, että kokeneemmille käyttäjille tulisi tarjota mahdollisuus tehdä haastavampia, monimutkaisempia projek-

teja. Viimeisenä alustan tulisi mahdollistaa monenlaisen sisällön luominen käyttäjän mieltymysten mukaan. (Burke & Kafai 2013.)

On olemassa pelintekoympäristöjä, jotka on joko suunniteltu nimenomaan opetuskäyttöön tai jotka muuten pyrkivät tarjoamaan käyttäjilleen helpon tavan luoda pelejä. Usein tällaiset ympäristöt perustuvat yksinkertaistettuun visuaaliseen ohjelmointikieleen tekstipohjaisen, tarkkaa syntaksia vaativan ”oikean” kielen sijaan. Taulukossa 1 esitellään joitain tällaisia ohjelmointikieliä. Matalaa lattiaa edustaa esimerkiksi visuaalinen, koodikomponenttien paikalleen raahaamiseen perustuva (drag-and-drop) ohjelmointi (kaikki taulukossa esitellyt), korkeaa kattoa oikean ohjelmointikielen esittely (Alice) ja leveitä seiniä mahdollisuus tehdä erityyppisiä projekteja (Alice, Scratch) (Burke & Kafai 2013). Planetoid Pioneersin osalta myös tähän palataan kohdassa 4.2.

Käytettävyydeltään hyvässä ohjelmistossa tulisi siis muistaa, kenelle ja millaisiin käyttötilanteisiin se on tarkoitettu. Tässä tapauksessa käyttö tapahtuu opetustilanteessa, käyttäjien ollessa opettajia ja oppilaita. Tällaisessa kontekstissa käytön aloittamisen tulisi olla helppoa ja nopeaa, sillä aika on aina rajattu. Luovassa opetusympäristössä helppouden lisäksi huomioon tulisi ottaa sillä rakennettavan sisällön joustavuus: kokeneempien käyttäjien tulee voida luoda monimutkaisempia projekteja, eikä ympäristö toisaalta saa haitata käyttäjiensä luovuutta asettamalla liian tiukkoja rajoja sillä luotavalle sisällölle. Planetoid Pioneersin suunnittelussa opetustilanteet eivät ole olleet päällimmäisenä mielessä, mutta tämä ei tarkoita, että se olisi täysin vailla hyviä puolia.

TAULUKKO 1. Visuaalisia ohjelmointikieliä (Resnick ym. 2009, 60, 63–65; AgentSheets Inc. 2014; Howland & Good 2015, 225, 227, 229; Carnegie Mellon University n.d.a, n.d.b; YoYo Games n.d.)

Nimi	Kehittäjä	Ominaisuudet
AgentSheets	AgentSheets Inc.	• Pelien ja simulaatioiden kehittämiseen • 2D-projektien teko • Visuaalinen, drag-and-drop-pohjainen ohjelmointi • Omien projektien jakaminen verkossa • ”Conversational programming” (”vuorovaikutteinen ohjelmointi”), jatkuva palaute käyttäjälle
Alice	Carnegie Mellon University (Looking Glass: Washington University in St. Louis)	• Pelien, animaatioiden ja interaktiivisten tarinoiden kehittämiseen • 3D-projektien teko • Palikoihin perustuva, visuaalinen ohjelmointikieli • Uusin versio tukee oikeaan Java-ohjelmointikieleen siirtymistä • Tämän pohjalta kehitetty vain animaatioiden ja tarinoiden luomiseen tarkoitettu Looking Glass (aiemmin Storytelling Alice)
Flip	University of Sussex (ei enää aktiivisesti tueta)	• Pelien kehittämiseen • 3D-projektien teko • Palikoihin perustuva, visuaalinen ohjelmointikieli • ”Bi-modaalinen” kieli, kääntää koodia dynaamisesti luettavalle, luonnolliselle kielelle • Lisäosa Never Winter Nights 2 -pelin kaupalliseen Electron Toolset -pelintekoympäristöön
GameMaker	YoYo Games	• Pelien kehittämiseen • 2D-projektien teko • Visuaalinen drag-and-drop-ohjelmointi • GameMaker Language (GML), yksinkertainen C-pohjainen kieli • Ei suunniteltu pelkästään opetuskäyttöön • Pelien julkaisu useille eri alustoille
Scratch	MIT Media Lab	• Pelien, animaatioiden, simulaatioiden ja tarinoiden kehittämiseen • 2D-projektien teko • Ohjelmointi visuaalisilla, legomaisilla palikoilla • Oman sisällön (kuvat, äänet) lisääminen tuotoksiin • Scratch-yhteisön rooli tärkeä, omien tuotosten jakaminen, muiden muokkaaminen

4 PLANETOID PIONEERS

4.1 Esittely

Planetoid Pioneers on kansainvälisen Data Realms -indiepelistudion kehittämä, vuonna 2018 julkaistu fysiikkapohjainen 2D-tietokonepeli-pelieditori. Se hyödyntää kehittäjien omaa Crush2D-pelimoottoria. Käyttäjien luoma sisältö on Planetoid Pioneersin keskiössä. Vuonna 2016 julkaistiin Contributor Edition -versio, joka sisältää jonkin verran pelattavaa sisältöä sekä editorin, jolla tämä on rakennettu. Version tarkoitus oli antaa yhteisölle mahdollisuus auttaa kehitysprosessissa luomalla peliin sisältöä ennen virallista julkaisua. Myös kaksi vuotta myöhemmin julkaistu valmis versio sisältää laajemman pelin lisäksi nämä työkalut. (Data Realms n.d.a, n.d.b; Steam 2018.) TG Desk -alustalla julkaistu versio on Contributor Edition build 6.

Kuten pelin nimi antaa ymmärtää, pelimaailma koostuu planetoideista. Kunkin planeetan voidaan ajatella vastaavan yhtä kenttää. Kuvassa 1 on osa Contributor Editioniin kuuluvasta esimerkkikentästä. Pelaaja voi koska vain siirtyä peli- ja editointitilojen välillä F2-näppäimellä. Kuvassa 2 on nähtävissä kuvan 1 kenttä editointitilassa. Crush2D-moottorille on rakennettu visuaalinen, verrattain helppokäyttöinen editori, jossa pelaajalla on käytettävissään sekä ne resurssit (tekstuurit, hahmot, esineet, materiaalit jne.), joilla olemassa olevat kentät on rakennettu, että mahdollisuus luoda täysin uutta sisältöä. Omien kuva- ja äänitiedostojen lisääminen onnistuu joko raahaamalla ne suoraan editori-ikkunan päälle tai lisäämällä ne resurssienhallinnan kautta haluttuun kansioon.

KUVA 1. Planetoid Pioneersin esimerkkikenttä pelitilassa (selkeyden vuoksi valaistus sekä tutkimattomat alueet piilottava *discovery map* on laitettu pois päältä)

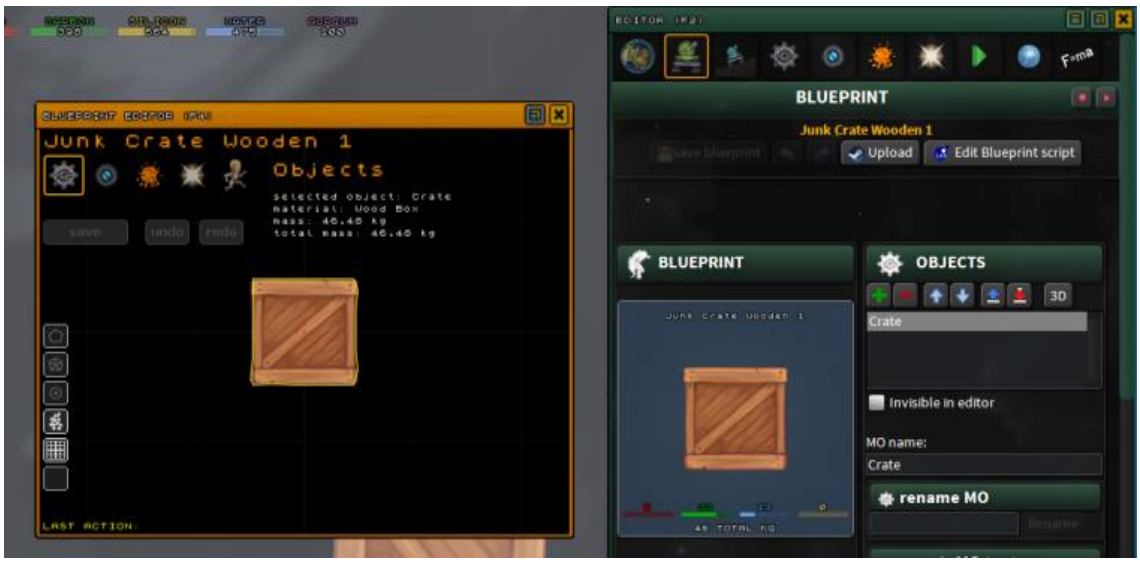
KUVA 2. Esimerkkikenttä editorissa

Fysiikkapohjaisuus on olennainen osa pelaamista. Tämä ilmenee niin hahmojen liikkeissä kuin peliobjektien tuhoutuvuudessakin. Liikkeeseen vaikuttavat esimerkiksi maasto, muut objektit sekä hahmon paino. Iso osa peliobjekteista on tuhottavissa esimerkiksi putoamalla niiden päälle, juoksemalla päin tai ampumalla. Tuhoutuvuuteen vaikuttaa materiaali (kuten metalli, puu, kivi), joka objektille on asetettu. Objekteja, joihin fysiikat vaikuttavat, kutsutaan editorissa liikuteltaviksi objekteiksi (engl. *movable objects*, *MO*).

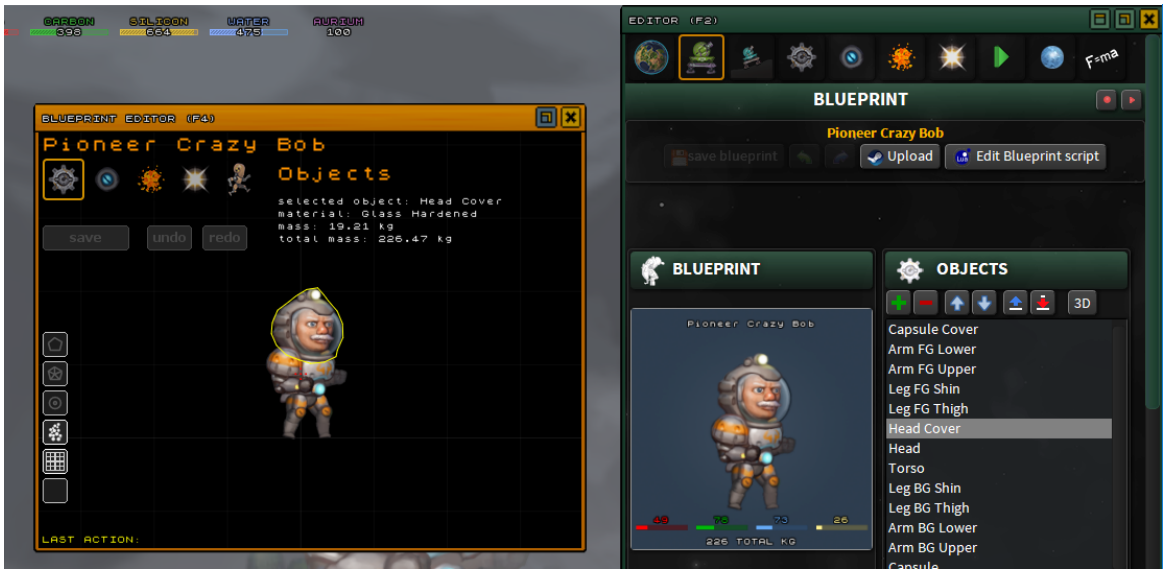
Kuten pelimoottoreissa yleensä, pelattava sisältö rakennetaan skenejen (engl. *scene*) sisälle. Skene voi esimerkiksi sisältää yhden kentän, Planetoid Pioneersin tapauksessa jokainen planetoidi on oma skenensä. Ne koostuvat staattisista maasto-objekteista (engl. *terrain objects*, *TO*) sekä toiminnallisista ”sinikopioiden” (engl. *blueprint*) instansseista. Maasto-objektit ovat liikkumattomia ja tuhoutumattomia, eli fysiikkamoottori ei vaikuta niihin. Ne voivat kuitenkin vaikuttaa hahmojen liikkeisiin muodollaan ja liukkaudellaan. Kuvan 3 valitut, keltaisella rajatut objektit ovat maasto-objekteja.

KUVA 3. Esimerkki maasto-objekteista

Sinikopiot puolestaan ovat useiden eri komponenttien yhdistelmiä. Ne koostuvat yhdestä tai useammasta liikuteltavasta objektista. Esimerkiksi niin yksinkertainen tuhoutuva laatikko (kuva 4) kuin monimutkaisempi pelattava hahmo (kuva 5) ovat sinikopioita. Sinikopioihin voi lisätä niiden toiminnallisuuteen vaikuttavia Lua-ohjelmointikielellä kirjoitettuja kooditiedostoja, skriptejä. Planetoid Pioneersiin on skriptien muokkaamista varten integroitu ZeroBrane Studio -ohjelmointiympäristö. Muita sinikopioihin liitettäviä komponentteja ovat nivelet, valonlähteet ja partikkeliefektit, näistä kerrotaan enemmän seuraavilla sivuilla. Sekä sinikopiot että skenet tallennetaan PNG-tiedostoina, jotka ovat helposti jaettavissa muille.



KUVA 4. Laatikko Blueprint-editorissa, oikealla Objects-paneelissa on nähtävissä sinikopion koostuvan yhdestä fysiikkaobjektista



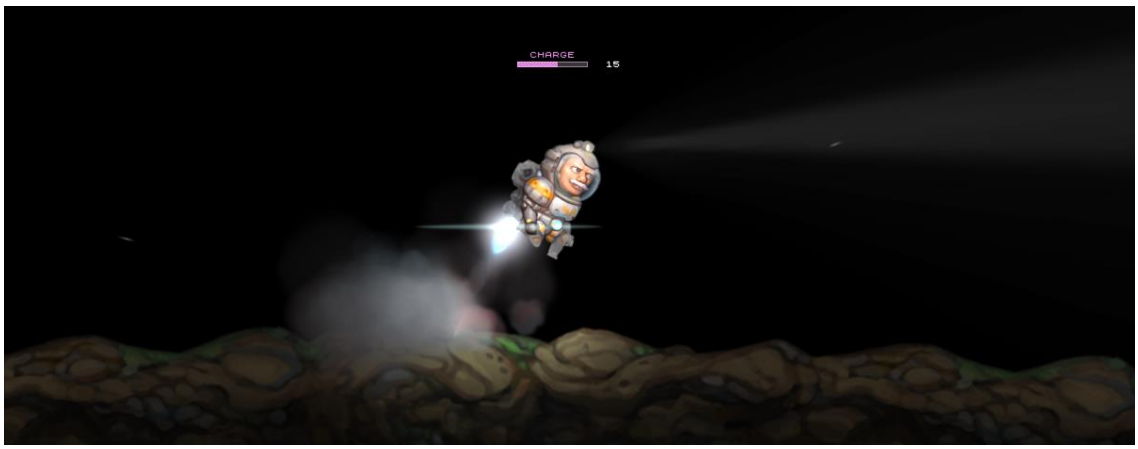
KUVA 5. Pelattava hahmo Blueprint-editorissa, Objects-paneelissa on nähtävissä sinikopion koostuvan useista fysiikkaobjekteista

Sinikopiota itseään ei voi tuoda skeneen, se on ennemminkin sapluuna, jonka pohjalta luoda sitä vastaavia kopioita eli instansseja. Editorissa näihin viitataan sanalla *assembly*. Esimerkiksi edellisen sivun kuvan 3 maasto-objektien yläpuolella näkyvä hahmo on sinikopion instanssi. Käytännössä tämä tarkoittaa, että yhden sinikopion pohjalta voidaan luoda loputon määrä instansseja ja kaikki sinikopiot ovat käytettävissä kaikissa skeneissä. Sinikopioon tallennettu muutos näkyy kaikissa sen pohjalta luoduissa kopioissa.

Objektit liitetään toisiinsa nivelillä (engl. *joint*). Ne voivat pyöriä nivelen luoman akselin ympäri käyttäjän asettamien rajoitteiden puitteissa. Esimerkiksi kitka ja ”jouselle” asetetut arvot vaikuttavat objektien liikkumiseen suhteessa toisiinsa. Kuvassa 6 vasemmalla hahmon päällä olevat valkoiset pisteet kuvaavat niveliä. Keskellä on 3D-näkymä, jossa näkyy, kuinka osat liittyvät toisiinsa. Tämä myös havainnollistaa, miltä näennäisen kaksiulotteiset hahmot näyttävät kolmiulotteisessa tilassa. Oikealla on ikkuna, jossa yksittäisiin niveliin vaikuttavia arvoja voidaan muokata.

KUVA 6. Hahmon koossa pitävät nivelet

Pelimoottori tukee lisäksi valonlähteiden ja partikkeliefektien lisäämistä. Esimerkiksi hahmoon on liitetty kaksi valonlähdettä: yksi selkeäreunainen otsalamppuun, toinen sumeampi hahmoon itseensä (kuva 7). Partikkeliefektit ovat pienistä fysiikkapohjaisista ”hiukkasista” koostuvia efektejä, joilla voidaan elävöittää peliä tai korvata yksinkertaisia, toistuvia animaatioita. Aiemmista kuvista jo tutuksi käyneen Bob-hahmon kursoria seuraavat silmät on esimerkiksi toteutettu partikkeliefekteillä. Myös muun muassa hahmon huuruava hengitys sekä rakettirepun liekit ja savu on luotu näin (kuva 7). Sinikopioiden ja skenejen tavoin partikkeliefektit tallennetaan PNG-tiedostoina.



KUVA 7. Hahmon valonlähteet ja rakettireppuun liitetty partikkeliefekti (pelitilassa)

4.2 Vahvuudet ja heikkoudet opetuskäytössä

Käyttäjien luoma sisältö on olennainen osa Planetoid Pioneersia ja editori on rakennettu niin, että myös ”ulkopuoliset” voivat käyttää sitä. Koska helppokäyttöisyys on ollut tekijöiden mielessä koko kehitysprosessin ajan, ainakin teoriassa ohjelmisto voisi taipua myös opetuskäyttöön. Erään kehittäjän Steam-sivuston keskustelupalstalla julkaiseman kirjoituksen mukaan Planetoid Pioneersia on itse asiassa käytetty kouluissa lupaavin tuloksin (Q-Tipps 2016). Tämä on tosin pelkkä sivuhuomautus, jolla korostetaan editorin helppokäyttöisyyttä. Tekstissä ei tarkemmin eritellä, millaisissa opetustilanteissa peliä on käytetty, minkä ikäisiä tai kuinka kokeneita oppilaat olivat.

Planetoid Pioneersin opetuskäytön suurin ongelma on, että vaikka se onkin suunniteltu helppokäyttöiseksi tietylle kohderyhmälle ja tiettyihin tilanteisiin, opettajat, oppilaat ja opetustilanteet eivät ole olleet prioriteettilistan kärkipäässä. Yksittäinen pelaaja, jolla on aikaa ja motivaatiota, voi hyvinkin oppia käyttämään editoria tehokkaasti. Opetustilanteissa aika on kuitenkin aina rajallinen ja vaikka monet tekijät voivatkin innostaa oppilaita (ks. alaluku 3.2), motivaatiokaan ei aina ole itsestäänselvyys (ks. alaluku 2.3). Näin ollen ohjelmiston käyttöönoton tulisi olla nopeaa eikä se saisi vaikeudellaan lannistaa käyttäjäänsä. Myös opettajalla olisi hyvä olla jonkinlainen käsitys ohjelmistosta eikä se hänen silmissään saisi vaikuttaa liian hankalalta luokkaympäristöön. Planetoid Pioneers pitää sisällään monipuolisen editorin, jonka kaikkien ominaisuuksien sisäistäminen vie aikansa.

Opittavuus ei siis ole luokkaympäristöä ajatellen paras mahdollinen. Nielsen määrittelee opittavuuden yksinkertaisten tehtävien suorittamisen helppoudeksi ensimmäisellä käyttökerralla (Nielsen 2012). Editori ei itsessään ohjeista käytön aloittamisessa, käyttäjä joutuu joko etenemään yrityksen ja erehdyksen kautta tai etsimään apua muualta. Ongelmia aiheuttaa myös ohjeistusten hajanaisuus. Tutoriaaleja löytyy esimerkiksi YouTube-videopalvelusta, Twitch-videostriimaussivustolta, wikeistä sekä Steam-yhteisösivuilta, tekijöitä ovat niin alkuperäiset kehittäjät kuin peliyhteisökin.

Aloittamisen hankaluudesta huolimatta editorin sisältö on loppujen lopuksi loogisesti jäsennelty. Kun tähän rakenteeseen tottuu, tulee käytöstä helpompaa. Visuaalisten painikkeiden lisäksi editori tarjoaa useita käyttöä vauhdittavia pikanäppäimiä. Tehokkuus viittaakin siihen, kuinka nopeasti käyttäjä kykenee suorittamaan tehtäviä sisäistettyään ohjelmiston käyttöliittymän (Nielsen 2012).

Muistettavuus tarkoittaa, kuinka helposti käyttäjä pystyy palaamaan ohjelman pariin tauon jälkeen (Nielsen 2012). Planetoid Pioneersin editori ei oleta käyttäjän muistavan kaikkea, useimmilla elementeille on selitys, joka tulee näkyviin kursorin ollessa niiden kohdalla. Lisäksi iso osa kulloinkin käytettävissä olevista pikanäppäimistä on näkyvissä ruudun vasemmassa yläkulmassa (nähtävissä kuvassa 2). Joidenkin pikanäppäimien selitykset on kuitenkin ”piilotettu” hieman paremmin, käyttäjä joutuu joko muistamaan ne itse tai etsimään ne Help-ohjeikkunasta.

Virheillä viitataan virheiden määrään, laatuun sekä niistä toipumiseen (Nielsen 2012). Assetteja muokatessa virheliikkeistä palautumista tukee melko mittava toimintojen kumoamismahdollisuus. Hieman suurempia ongelmia saattaa syntyä tallennusvaiheessa, sillä ohjelma tallentaa muutokset oletusarvoisesti olemassa olevan tiedoston päälle kysymättä käyttäjältä varmennusta. Koska esimerkiksi sinikopioon tallennettu muutos heijastuu kaikissa sen instansseissa, muokkaamisella voi olla mittava vaikutus useisiin skeneihin. Skeneistä, sinikopiosta ja partikkeliefekteistä luodaan varmuuskopiota, joilla tämänkaltaisista tilanteista voidaan palautua, mutta kokematon käyttäjä ei välttämättä osaa etsiä niitä.

Vakavampi ongelma on, että ohjelman voi saada täysin jumiin. Editori pyörittää oletusarvoisesti simulaatiota (esim. fysiikoiden vaikutus, skriptien ajo) editointitilassa. Tässä on etunsa, sillä käyttäjä näkee välittömästi tekemiensä muutosten vaikutukset. Ongelmia syntyy, jos ajettava skripti on jotenkin ”rikkinäinen”. Päättymätön silmukka koodissa

saa minkä tahansa ohjelman kaatumaan, mutta Planetoid Pioneersin kohdalla tämä voi osoittautua tavallistakin hankalammaksi. Mikäli viallinen skripti on jo tallentunut osaksi PNG-tiedostoa, kooditiedoston muokkaaminen ei korjaa tilannetta, sillä käynnistettäessä editori korvaa sen sisällön PNG:hen liitetyllä tiedolla. Näin ollen se alkaa välittömästi ajaa viallista koodia ja kaatuu jälleen. Tätä ongelmaa ei ole mahdollista korjata editorissa, vaan käyttäjä joutuu korvaamaan PNG-tiedoston varmuuskopiolla resurssienhallinnan kautta. Mikäli varmuuskopiota ei syystä tai toisesta ole, on täydellinen toipuminen erittäin hankalaa, ellei mahdotonta.

Tyytyväisyys tarkoittaa sitä, kuinka miellyttävää ohjelmiston käyttö on (Nielsen 2012). Sitä ei tietenkään voi tässä tilanteessa kovinkaan laajasti ja objektiivisesti arvioida. Positiivisesti käyttökokemukseen voivat vaikuttaa sisällön selkeä järjestely, käytön joustavuus, tiettyjen ominaisuuksien helppous sekä kontekstiherkät ohjetekstit. Vaikka käyttöliittymä voikin alkuun tuntua sekavalta, ominaisuuksien jaottelussa on logiikkansa. Eritasoisille käyttäjille on tarjolla vaihtoehtoja ja omien tiedostojen lisääminen on helppoa. Editori ei oleta käyttäjän muistavan kaikkea, vaan näyttää selityksen miltei kaikelle kursorin ollessa kohdalla. Negatiivista on ennen kaikkea editorin monimutkaisuus sekä vakavien virhetilanteiden mahdollisuus. Vasta-alkaja tuskin pystyy käyttämään editoria ilman ulkoista apua. Vaikka virheistä onkin usein mahdollista toipua, jotkin niistä voivat olla erittäin vakavia. Pahimmillaan ohjelman jumiutuminen saattaa melkeinpä pakottaa lopettamaan sen käytön. Lyhyesti sanottuna Planetoid Pioneersin käytettävyydessä opetustilanteessa on parantamisen varaa, vaikka hyviä puolia onkin.

Matalien lattioiden, korkeiden kattojen ja leveiden seinien suhteen Planetoid Pioneers ei ole suoraan verrattavissa aiemmin esiteltyihin visuaalisiin ohjelmointikieliin, tässä tapauksessahan puhutaan koko ohjelmasta eikä pelkästään ohjelmointiosuudesta. Se ei myöskään tarjoa käyttäjilleen helpompaa versiota oikeasta ohjelmointikielestä eikä sitä, toisin kuin useimmat taulukossa 1 esitellyistä, ole suunniteltu opetuskäyttöön. Ohjelmointiin keskittymisen sijaan seuraavaksi arvioidaan siis koko ohjelman käytön aloittamisen helppoutta, sen tarjoamia mahdollisuuksia edistyneemmälle käyttäjälle sekä sillä luotavan sisällön monimuotoisuutta.

Editori tarjoaa paljon valmiita rakennuspaloja. Näin ollen skeneä rakentaessa ei tarvitse osata ohjelmoida lainkaan tai muutenkaan käyttää monimutkaisempia ominaisuuksia. Halutut osaset voidaan helposti raahata osaksi skeneä. Käytön aloituksessa auttavat

myös näkyvät painikkeet ja kontekstiherkät ohjetekstit. Uusi käyttäjä ei kuitenkaan välttämättä löydä näitä rakennuspaloja editorin monimutkaisuudesta johtuen. Lua-ohjelmointikieltä ei esitellä, vaan oletetaan, että mikäli haluaa kirjoittaa skriptejä, tutustuu siihen omatoimisesti. Kokeneemmat käyttäjät voivat tehdä monimutkaisemmin muutoksia tai luoda kokonaan uusia sinikopioita. Halutessaan he voivat muokata ja lisätä skriptejä. Kokemuksen kasvaessa käyttö tehostuu ja näkyvistä painikkeista voidaan siirtyä pikanäppäimiin. Korkeiden kattojen kriteerin editori täyttää siis helposti.

Koska käyttäjät voivat vapaasti muokata olemassa olevia palasia ja tuoda editoriin omia tiedostojaan, mahdollisuudet erilaisen sisällön luomiseen ovat teoriassa rajattomat. Sisällön tyyppi on kuitenkin melko tarkkaan ennalta määrätty. Vaikka kekseliäämpi käyttäjä saattaisikin onnistua luomaan muunkinlaisia projekteja, Planetoid Pioneers on tarkoitettu pelieditoriksi. Myös planetoideista koostuva pelimaailma ja fysiikkapohjaisuus asettavat omat rajoitteensa.

Yksi tärkeä etu, jota ei ole vielä juurikaan käsitelty, on Planetoid Pioneersin yhteisöllisyys. Vuorovaikutus on olennainen osa konstruktionistista opetusteoriaa ja esimerkiksi monien aihetta käsittelevien julkaisujen takana olevat Kafai ja Burke pitivät yhteisöjä niin tärkeinä, että jatkoivat tietokoneajattelun termiä omallaan, joka kattaa myös niihin osallistumisen. Kuten Planetoid Pioneersin esittelyluvussa mainittiin, peliyhteisö otettiin mukaan jo ennen virallista julkaisua tarjoamalla mahdollisuus tehdä sisältöä, joka saattaisi päästä osaksi lopullista peliä. Pelaajien osallistuminen sisällön luontiin on olennainen osa myös valmista versiota.

Planetoid Pioneers on kehitetty niin, että käyttäjät voivat helposti jakaa tekemäänsä sisältöä eteenpäin. Tuotokset tallennetaan PNG-tiedostoina, joiden siirtäminen koneelta toiselle on yksinkertaista. Oppilailla on myös mahdollisuus osallistua laajempaan yhteisöön jakamalla tuotoksiaan internetin kautta suuremmalle yleisölle. Muiden luoman sisällön jatkokehittäminen on mahdollista ja näin prosessista tulee entistä osallistavampi. Lisäksi editorilla rakennettu sisältö tukee paikallista moninpeliä, yhteensä neljä pelaajaa voi liittyä samaan peliin.

Planetoid Pioneersin opetuskäytöllä on sekä hyvät että huonot puolensa. Ensikosketus editoriin voi olla hämmentävä, mutta kun sitä oppii navigoimaan, melko pienellä vaivalla on mahdollista luoda kokonaisia skenejä. Ohjelmointitaitoja ei välttämättä tarvita lainkaan. Kokemuksen kasvaessa käyttö tehostuu ja käyttäjällä on mahdollisuus luoda

niin monimutkaisia projekteja kuin mielikuvituksensa vain sallii. Vaikka pelimoottori asettaakin melko tiukat rajat sille, millaisia projekteja sen päälle voidaan rakentaa, annetuissa puitteissa räätälöintimahdollisuudet ovat teoriassa rajattomat. Omien tuotosten jakaminen sekä muiden luomien assettien jatkokehittäminen on helppoa. Vakavien virheiden mahdollisuus on kuitenkin olemassa. Ongelmallista on myös, ettei editori tue ohjelmointitaitojen kehityksessä.

5 OHJEISTUKSEN TOTEUTUS

5.1 Toteutustapa

Ohjeistuksen toteutustavaksi valittiin kirjallinen PDF-dokumentti. Tähän päädyttiin laatijan omien taitojen ja valitun muodon havainnollisuuden perusteella. Muita toimeksiantajan ehdottamia vaihtoehtoja olivat videot ja ThingLink-sivuston avulla luodut interaktiiviset kuvat. Videot olisivat olleet näistä kenties havainnollisin, mutta työn tekijällä ei ole lainkaan kokemusta niiden editoinnista. Kuvat taas, nimensä mukaisesti, olisivat olleet PDF-ohjeistusta interaktiivisempia, mutta ne olisivat suppean esitystapansa takia tarjonneet vain pinnallisen katsauksen editorin sisältöön. Alkuperäiseen suunnitelmaan itse asiassa kuului tällaisten kuvien tekeminen laajemman ohjeistuksen rinnalle, mutta ajanpuutteen vuoksi ne jätettiin lopullisen työn ulkopuolelle.

PDF-dokumentti koottiin Adoben InDesign-julkaisuohjelmalla. Ohjelma valittiin sen tarjoamien joustavien tekstin ja kuvien asettelu- ja muotoilumahdollisuuksien vuoksi. Ohjeistuksen sisältö kerättiin perehtymällä editoriin olemassa olevien verkkotutoriaalien avustamana. 5 More Minutesin asiakaskunta on kansainvälinen, näin ollen ohjeistus kirjoitettiin englanniksi.

Dokumentissa huomioon tuli ottaa ennen kaikkea sen kohdeyleisö eli yläasteikäiset oppilaat, joilla ei välttämättä ole kokemusta tämänkaltaisista ohjelmista, sekä käyttötilanne. Tekstiä ei saanut olla liikaa, sillä lukeminen vie aikaa, eikä välttämättä kiinnosta oppilaita. Esiteltyjä asioita tuli havainnollistaa kuvin ja esimerkein, ohjeistuksen sisältö tuli jäsennellä mahdollisimman selkeästi ja loogisesti. Kohdeyleisön ikä ja koulutusaste rajattiin toimeksiantajalta saadun arvion perusteella.

5.2 Lopputuloksen esittely

Valmis ohjeistus on 48-sivuinen PDF-dokumentti. Olennaisin sisältö jakautuu kuuteen pääotsikkoon, jotka lähtevät liikkeelle pelaamisen esittelystä ja etenevät staattisten ympäristöjen rakentamisen kautta interaktiivisemman sisällön luontiin. Taulukossa 2 on lueteltu nämä pääotsikot ja niiden sisältö tiivistetysti. Lisäksi dokumentti sisältää lyhyen

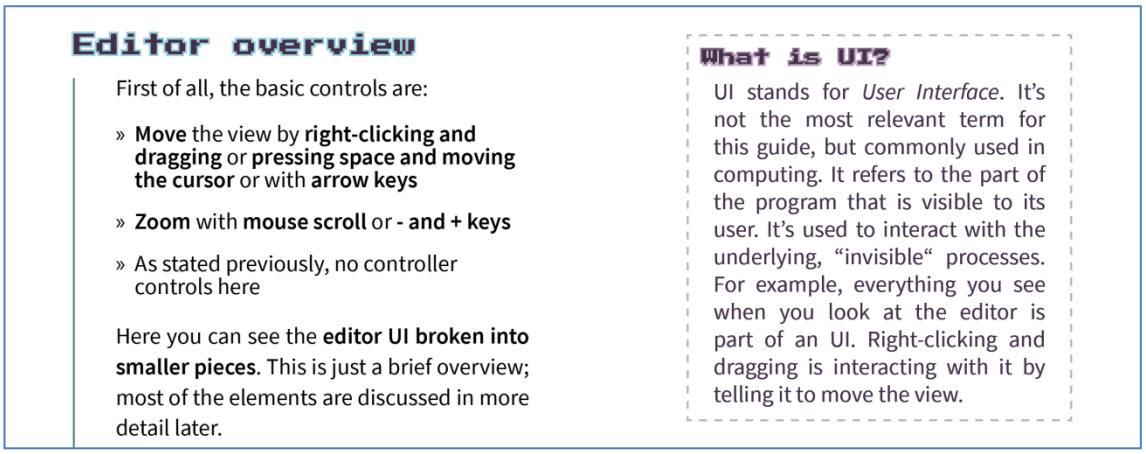
johdannon, tiedostojen jakoa käsittelevän sivun sekä lähdeluettelon. Ohjeistus on kokonaisuudessaan luettavissa raportin liitteenä.

TAULUKKO 2. Ohjeistuksen sisältö tiivistetysti

Otsikko	**Sisältö**
Welcome to Planetoid Pioneers	Pelin ja sen kontrollien lyhyt esittely
Getting to know the editor	Editorinäkymään siirtyminen, käyttöliittymän elementtien esittely
Building a game world	Planetoidin muokkaaminen staattisilla maastoobjekteilla
Adding interactivity	Sinikopioinstanssien lisääminen pelimaailmaan, yhteyksien lisääminen näiden välille
Customising blueprints	Olemassa olevien sinikopioiden muokkaaminen, liikuteltavien objektien ja nivelien lisääminen ja muokkaaminen
Particles and lights	Sinikopioihin liitettyjen partikkelien ja valojen muokkaaminen, partikkelitiedostojen luominen, partikkelien ja valojen lisääminen pelimaailmaan

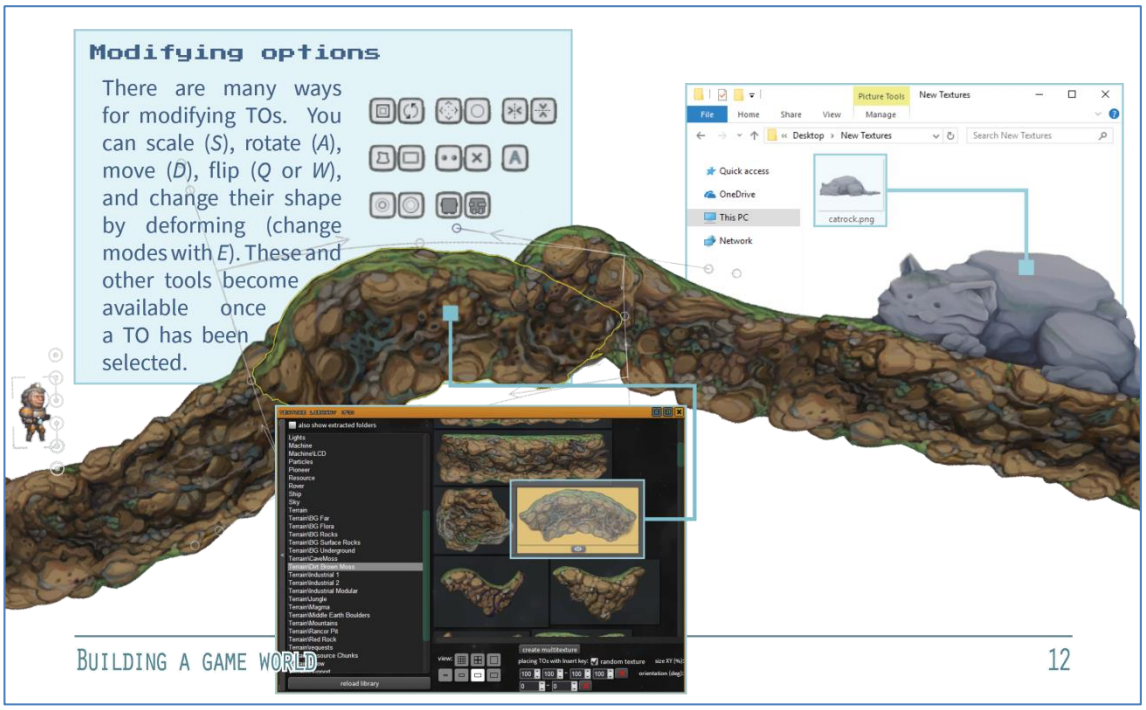
Taulukossa esitellyt pääotsikot jakautuvat pienempiin alaotsikoihin (niitä ei tässä listata niiden suuren määrän vuoksi, mutta ne ovat nähtävissä ohjeistuksen sisällysluettelossa). Valtaosa tekstisisällöstä on kirjoitettu näiden alaotsikoiden alle. Lisäksi joitain pienempiä, kutakin aihetta sivuavia asiakokonaisuuksia käsitellään lyhyesti muusta tekstistä irrallisissa tekstilaatikoissa.

Sisältö on värikoodattu kolmella värillä: sininen, violetti ja keltainen. Ne valittiin Planetoid Pioneersin logossa ja käyttöliittymässä käytettyjen värien perusteella. Sininen on dokumentin pääväri, sillä otsikoidut osiot ovat editorin ymmärtämisen kannalta tärkeimpiä. Violetti puolestaan viittaa täydentävään tietoon tai haastavampiin esimerkkeihin. Monimutkaisimpien osioiden (taulukon 2 neljä viimeistä) jäljestä löytyvät keltaisella värikoodatut Troubleshooting-sivut, joilla on lueteltu joitain mahdollisia ongelmia sekä niiden ratkaisuja. Kuvassa 8 on nähtävissä esimerkki käytetystä värikoodauksesta. Olennaisimpia kohtia on korostettu lihavoinnilla, jotta tekstin nopea silmäily olisi helpompaa.



KUVA 8. Esimerkki sinisellä värikoodatusta "tärkeästä" tekstistä ja violetista, täydentävästä tekstilaatikosta (ohjeistus s. 9)

Useimmilla sivuilla on tekstin lisäksi kuvia. Kuvat sekä havainnollistavat tekstissä esiteltyjä asioita että elävöittävät dokumenttia. Valtaosa kuvista on Planetoid Pioneersista otettuja kuvakaappauksia. Osa on muokkaamattomia, neliskulmaisia kuvakaappauksia, toisia on hieman editoitu esimerkiksi poistamalla taustat. Tämä antoi enemmän mahdollisuuksia niiden asettelulle ja sai dokumentin näyttämään elävämmältä. Kuvassa 9 on esimerkki, jossa on käytetty sekä muokkaamattomia kuvakaappauksia ikkunoista että kuvia, joiden tausta on poistettu. Ohjeistusta varten on piirretty jonkin verran uutta grafiikkaa, jota käytetään esimerkeissä, joissa omia tekstuureja tuodaan editoriin. Näistäkin valtaosa on tosin itse dokumentissa nähtävissä vain osana kuvakaappauksia. Esimerkiksi kuvan 9 oikean reunan kivi on ohjeistuksen havainnollistamista varten luotua grafiikkaa.



KUVA 9. Esimerkki tavoista, joilla kuvia on lisätty tekstin tueksi (ohjeistus s. 12)

Ohjeistuksen sivun koko vastaa A4-kokoista paperia. Se on toteutettu olettamuksella, että sitä selataan tietokoneella, eikä esimerkiksi tulosteta avattavan kirjan muotoon. Tulostusmahdollisuutta ei ole huomioitu myöskään värien käytössä tai kuvien määrässä, dokumentti on pitkä ja sisältää paljon molempia. Kouluissa ei välttämättä oltaisi valmiita tulostamaan sitä sellaisenaan kaikille oppilaille. Tärkeä tavoite kuitenkin oli havainnollisen ja mielenkiintoisen oppaan kokoaminen, värien ja kuvien käytön rajoittaminen olisi haitannut tätä.

Tekstisisältö pyrittiin pitämään mahdollisimman tiiviinä ja asettelemaan yhdessä kuvien kanssa sivuille selkeästi ja miellyttävästi. Kumpikaan tehtävistä ei ollut helppo: Planetoid Pioneers on monimutkainen ohjelma, jonka selittäminen ei aina onnistu lyhytsanaisesti. Toisaalta tekstin määrä ja rakenne vaikuttivat aina myös kuvien kokoon ja asetteluun. Dokumentin sisällön ja taittamisen suunnittelu ja toteutus olivatkin koko työn aikaa vievin osuus.

6 POHDINTA

Videopeleillä on niiden perusrakenteen ansiosta ainakin teoriassa potentiaalia olla merkittäviä opetuksen apuvälineitä. Käytännössä tätä potentiaalia ei kuitenkaan ole onnistuttu alan tutkimusten hajanaisuudesta johtuen kiistatta todistamaan. Tästä huolimatta videopelien on havaittu kasvattavan oppilaiden motivaatiota osallistua opetukseen. Sama motivaatio voi olla läsnä myös pelinteossa. Pelinteko käy käsi kädessä nykymaailmassa vallitsevan laajemman mediakulttuurin kanssa: itse tekeminen ja osallistuminen ovat monille osa arkea. Sillä on potentiaalia opettaa monia muuttuvassa maailmassa tärkeitä taitoja, jotka eivät rajoitu pelkästään ohjelmointiin ja teknologian käyttöön. Pelintekoon liittyy esimerkiksi luovuutta, ongelmanratkaisua ja tiimityötä.

Työn toimeksiantaja on 5 More Minutes -opetuspeliyritys, jonka alustalla on julkaistu Planetoid Pioneers -peli-pelieditori. Kuten tämän raportin teoriaosuus osoittaa, pelaamisen rinnalla pelinteolla on paikkansa opetuksessa ja Planetoid Pioneersin kaltainen ympäristö voisikin olla tärkeä osa yrityksen tarjontaa. Kyseistä ohjelmistoa ei kuitenkaan ole suunniteltu opetuskäyttöön, mikä näkyy sen monimutkaisuudessa. Käytön aloittamisen helpottamiseksi laadittiin aloitusopas, jota 5 More Minutes voi jaella asiakkailleen Planetoid Pioneersin rinnalla.

Opinnäytetyön tuloksena syntyi 48-sivuinen englanninkielinen PDF-opas. Oppaan tärkein sisältö jakautuu kuuteen pääotsikkoon ja pienempiin alaotsikoihin. Valtaosa tekstistä on kirjoitettu näiden alaotsikoiden alle, joitain irrallisempia kokonaisuuksia lisättiin sivuille erillisissä tekstilaatikoissa. Tekstiä on havainnollistettu ja elävöitetty kuvin, joista valtaosa on editorista otettuja kuvakaappauksia.

Ohjeistuksen laatiminen osoittautui odotettua haastavammaksi ja vei huomattavasti arvioitua enemmän aikaa. Toteutuksessa piti ottaa asiasisällön lisäksi huomioon sen esitystapa. Vaikka Planetoid Pioneersin editori on monimutkainen ohjelma, sen käyttöä piti pyrkiä esittelemään mahdollisimman tiiviisti ja yksinkertaisesti. Tämän lisäksi ohjeistus tuli yrittää pitää mahdollisimman mielenkiintoisena ja tekstisisältöä tuli havainnollistaa kuvin. Sekä kuvat että teksti piti asetella sivuille mahdollisimman selkeästi ilman, että ne tuntuivat liian tyhjiltä tai täyteen ahdetuilta. Opinnäytetyön tekijällä ei ollut aiempaa kokemusta vastaavanlaisen dokumentin taittamisesta.

Työn ensisijaisena tarkoituksena oli Planetoid Pioneers -aloitusoppaan laatiminen. Tähän päämäärään päästiin, joten työtä voitaneen pitää näiltä osin onnistuneena. Ohjeistuksen tekemiseen kului kuitenkin paljon suunniteltua enemmän aikaa, joten joitain osakokonaisuuksia jouduttiin jättämään pois. Näistä merkittävin oli ohjeistuksen ohjelmointia käsittelevä osuus. Osa alkuperäistä suunnitelmaa oli Lua-skriptauksen perusteiden esittely. Prosessin loppua kohden kävi kuitenkin selväksi, että tämä olisi ollut liian suuri kokonaisuus käsiteltäväksi jäljellä olevassa ajassa. Toinen puuttuva kokonaisuus ovat jo aiemmin mainitut ThingLink-kuvat. Suunnitelmaan kuului lisäksi ohjeistuksen testauttaminen toimeksiantajayrityksen kautta iältään kohderyhmää vastaavilla nuorilla. Tämä olisi toki ollut tärkeää ohjeistuksen parantelun kannalta sekä mahdollisesti tarjonnut tilaisuuden vertailla testauksen tuloksia teoriaosuudessa esiteltyihin tutkimuksiin, mutta myös tästä suunnitelmasta jouduttiin ajanpuutteen vuoksi luopumaan.

Näin ollen ilmeisin jatkokehitysehdotus on joko tämän dokumentin jatkaminen ohjelmointia käsittelevällä osiolla tai kokonaan uuden, vain skriptaukseen keskittyvän ohjeistuksen luominen. Myös pois jääneet ohjelmiston käyttöliittymää suppeammin esittelevät interaktiiviset kuvat sekä ohjeistuksen testauttaminen ja muutosten tekeminen tulosten perusteella ovat jatkokehitysmahdollisuuksia. Lisäksi dokumentti voitaisiin kääntää muille kielille, ensimmäiseksi luultavasti suomeksi. Hieman erilainen, ohjeistukseen suoraan liittymätön, mahdollisuus on uusien TG-assettien luominen Planetoid Pioneersiin.

Vaikka työn teoriaosuus antaakin myönteisen kuvan videopelien opetuskäytöstä, tässä vaiheessa lienee aiheellista huomauttaa, että käytettyjen lähteiden kirjoittajat ovat aiheen puolestapuhujia. Vaikka niistä useissa viitattiinkin muiden tutkijoiden julkaisuihin ja joissain sivuttiin alan kohtaamia haasteita, olivat niiden kirjoittajat puolueellisia. Negatiivisten löydösten syitä ei välttämättä juurikaan pohdittu. Esimerkiksi tutkimus, jonka mukaan pelinteko ei motivoinut tyttöjä tavoittelemaan uraa STEM-aloilla kuitattiin melko pinnallisella toteamuksella alan suhtautumisesta naisiin. Lisäksi varsinkin konstruktionismia käsittelevissä lähteissä samat nimet toistuivat useasti. Teorian puolesta aihetta voitaisiin tutkia laajemmasta ja objektiivisemmasta näkökulmasta, käyttäen lähteinä myös julkaisuja, jotka suhtautuvat pelien opetuskäyttöön varauksella.

LÄHTEET

AgentSheets Inc. 2014. What is AgentSheets?. Päivitetty 2014. Luettu 16.9.2018. http://www.agentsheets.com/products/index.html

Baytak, A. & Land, S. M. 2010. A case study of educational game design by kids and for kids. Procedia Social and Behavioral Sciences. 2 (2), 5242–5246.

Burke, Q. & Kafai, Y. 2013. A Decade of Game-Making for Learning. From Tools to Communities. Julkaistu tammikuussa 2014. Verkkojulkaisu teoksesta Agius, H. & Angelides, M. C. (toim) Handbook of Digital Games. Tulostettu 8.8.2018. https://www.researchgate.net/publication/305426258_A_decade_of_programming_games_for_learning_From_tools_to_communities

Carnegie Mellon University. N.d.a. Alice-kotisivu. Luettu 16.9.2018. https://www.alice.org/

Carnegie Mellon Univeristy. N.d.b. Our History. Luettu 16.9.2018. https://www.alice.org/about/our-history/

Data Realms. N.d.a. Data Realms. Luettu 19.9.2018. http://www.datarealms.com/press/index.php

Data Realms. N.d.b. Planetoid Pioneers. Luettu 19.9.2018. http://www.datarealms.com/press/sheet.php?p=planetoid_pioneers

de Freitas, S. 2018. Are Games Effective Learning Tools?. A Review of Educational Games. Educational Technology & Society. 21 (2), 74–84.

de Paula, B. H., Burn, A., Noss, R. & Valente, J. A. 2018. Playing Beowulf. Bridging computational thinking, arts and literature through game-making. International Journal of Child-Computer Interaction. 16 (kesäkuu), 39–46.

Filmora. 2018. Digital Video Game Trends and Stats for 2018. Julkaistu 17.2.2018. Luettu 24.10.2018. https://filmora.wondershare.com/infographic/video-game-trends-and-stats.html

Gilbert, B. 2018. 'Minecraft' is still one of the biggest games in the world, with nearly 75 million people playing monthly. Julkaistu 21.1.2018. Business Insider Nordic. Luettu 16.9.2018. https://nordic.businessinsider.com/minecraft-has-74-million-monthly-players-2018-1

Howland, K. & Good, J. 2015. Learning to communicate computationally with Flip. A bi-modal programming language for game creation. Computers & Education. 80 (tammikuu), 224–240.

ISO 9241-11:2018. 2018. Usability. Definitions and concepts. Geneve: International Organization for Standardization. Luettu 16.9.2018. https://www.iso.org/obp/ui/#iso:std:iso:9241:-11:ed-2:v1:en

Kafai, Y. B. 2006. Playing and Making Games for Learning. Instructionist and Constructionist Perspectives for Game Studies. Games and Culture. 1 (1), 36–40.

Kafai, Y. B. 2009. Constructionism. Teoksessa Sawyer, R. K. (toim.) The Cambridge handbook of the learning sciences. New York: Cambridge University Press, 35–46.

Kafai, Y. B. 2016. From Computational Thinking to Computational Participation in K-12 Education. Communications of the ACM. 59 (3), 26–27.

Kafai, Y. B. & Burke, Q. 2015. Constructionist Gaming. Understanding the Benefits of Making Games for Learning. Educational Psychologist. 50 (4), 313–334.

Koop, A. 2018. Gamification is Just Icing on a Lousy Cake. Luento. Nordic Game 2018 23.–25.5.2018. Malmö.

Nielsen, J. 2012. Usability 101. Introduction to Usability. Julkaistu 4.1.2012. Luettu 16.9.2018. https://www.nngroup.com/articles/usability-101-introduction-to-usability/

Partnership for 21st Century Learning. 2015. P21 Framework Definitions. Washington.

Q-Tipps. Planetoid Pioneers -kehittäjä. 2016. FAQ. Planetoid Pioneers Contributor Edition. Steam-keskustelu. Julkaistu 10.4.2016. Päivitetty 18.4.2016. Luettu 27.9.2018. https://steamcommunity.com/app/300260/discussions/0/365163686073914566/

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B. & Kafai, Y. 2009. Scratch. Programming for All. Communications of the ACM. 52 (11), 60–67.

Statista. 2018. Value of the global video games market from 2012 to 2021 (in billion U.S. dollars). Luettu 19.9.2018. https://www.statista.com/statistics/246888/value-of-the-global-video-game-market/

Steam. 2018. Planetoid Pioneers. Luettu 25.9.2018. https://store.steampowered.com/app/300260/Planetoid_Pioneers/

SuperData Research. 2018. 2017 year in review. Digital games and interactive media. New York.

Turkay, S. & Adinolf, S. 2012. What do players (think they) learn in games? Procedia Social and Behavioral Sciences. 46 (2012), 3345–3349.

WePC. 2018. 2018 Video Game Industry Statistics, Trends & Data. Päivitetty toukokuussa 2018. Luettu 24.10.2018. https://www.wepc.com/news/video-game-statistics/#gamers-demographic

Whitton, N. 2014. Digital Games and Learning. Research and Theory. 1. painos. New York: Routledge.

Wing, J. M. 2006 Computational Thinking. Communications of the ACM. 49 (3), 33–35.

YoYo Games. N.d. Gamemaker for Education. Luettu 16.9.2018. https://www.yoyogames.com/education

LIITTEET

Liite 1. Planetoid Pioneers – Getting Started -opas

Contents

About this guide

The purpose of this guide is to help you get started with creating content using *Planetoid Pioneers*' editor. The contents are colour coded with three colours: blue, purple, and yellow. Contents under **blue** headlines (like the one above) and in blue boxes are the **most important ones** as they go through the basics of how to use the editor. The ones with **purple** headers or outlined with purple boxes offer **additional, related information** and **more complex tasks**. After some of the sections there are *Troubleshooting* pages colour coded with **yellow**. These list some of the **problems you might face and possible solutions to them**.

This guide was compiled using *Planetoid Pioneers Contributor Edition* build 6 on a Windows PC. The author of this guide is in no way associated with *Data Realms*, the creators behind *Planetoid Pioneers*.

YOU'VE JUST STARTED THE GAME FOR THE FIRST TIME AND CRASH-LANDED ON YOUR FIRST PLANETOID. THIS IS A TUTORIAL PLANETOID WHERE YOU CAN FAMILIARISE YOURSELF WITH THE GAMEPLAY BEFORE YOU START CREATING YOUR OWN CONTENT.

Playing the game

There are two things central to *Planetoid Pioneers*' gameplay: first, it is heavily **physics-based**. Second, almost anything that can be moved is also **destructible** and can be **atomised** and **reassembled**.

At this point you're probably better off testing the game yourself to see how everything works. However, here are some things you might want to know:

» The game can be played with a **keyboard and mouse** or an **Xbox controller**

» Bring up the ***Help* window** by pressing ***Esc* or *F1*** key on the keyboard or ***Start*** button on the controller; there you can view the controls (top) and a tutorial (bottom)

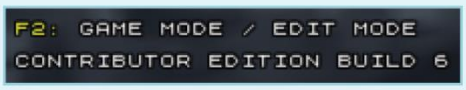
» The game supports local **multiplayer** for up to 4 players

Mouse & keyboard controls

LMB/RMB = left/right mouse button

A, D	Move left and right
W	Jump
S	Crouch
Mouse	Aim
LMB	Atomise object
RMB	View blueprints
E	Pick up/drop item
LMB or RMB	Use item
Q	Put away item
F	Enter/exit vehicle
Left shift	Brake vehicle
Mouse scroll or + & -	Zoom
F9	Restart
X	Open/close Planetoid map (WASD/RMB + drag: move)
P	Pause/continue

Different modes

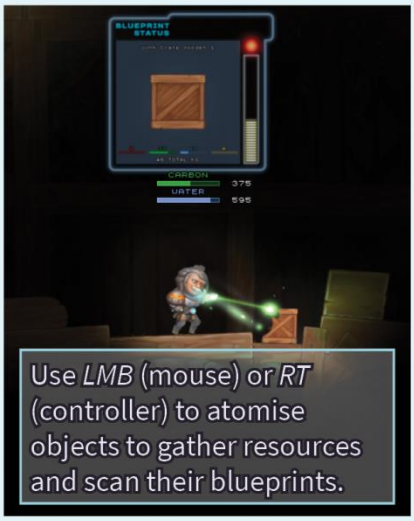


You may have spotted this text on the upper left corner. Playing the game happens in game mode and assuming you haven't pressed *F2*, that's where you currently are. Edit mode will be the main focus of this guide; however, if you want to keep playing for now, stay in game mode. If not, move on to the next section of this guide.

Controller controls

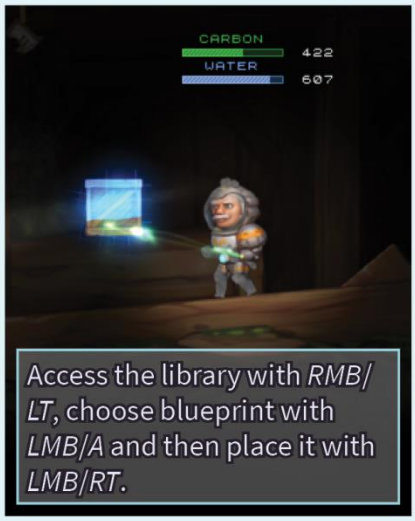
L stick	Move
A	Jump
B or LB	Crouch
R stick	Aim
RT	Atomise object
LT	View blueprints
X	Pick up/drop item
RT	Use item
Y	Put away item
B	Enter/exit vehicle
RB	Brake vehicle
D-pad up & down	Zoom
Back	Restart

Atomising & assembling



Use *LMB* (mouse) or *RT* (controller) to atomise objects to gather resources and scan their blueprints.

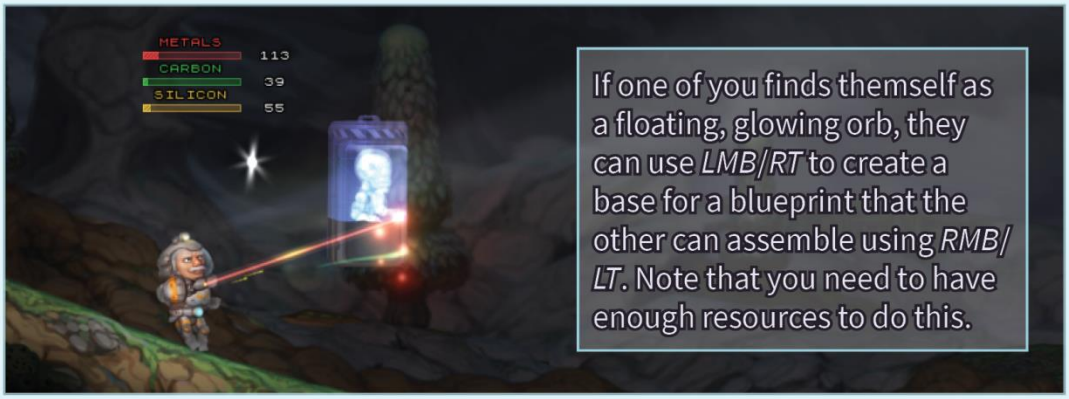
Once you have fully scanned an object, it is added to your blueprint library. Now you can create copies of it.



Access the library with *RMB/LT*, choose blueprint with *LMB/A* and then place it with *LMB/RT*.

Playing with others

The game supports local multiplayer for up to 4 players. If you have Xbox controllers available, your friends can simply plug them in and join the game any time (while in game mode).



If one of you finds themself as a floating, glowing orb, they can use *LMB/RT* to create a base for a blueprint that the other can assemble using *RMB/LT*. Note that you need to have enough resources to do this.

About this version

The *Planetoid Pioneers* version you're using is made with content creation in mind. This means that, even though there are some playable levels, the main objective is to create your own content. This guide introduces the tools available to help you get started with your own creations.

NOW THAT YOU'VE SEEN HOW THE GAME WORKS, IT'S TIME TO START EXPLORING THE EDIT MODE. IT CAN SEEM A BIT OVERWHELMING AT FIRST, BUT LUCKILY YOU HAVE THIS GUIDE TO HELP. IN THIS SECTION YOU'LL BE BRIEFLY INTRODUCED TO EVERYTHING YOU SEE IN EDIT MODE.

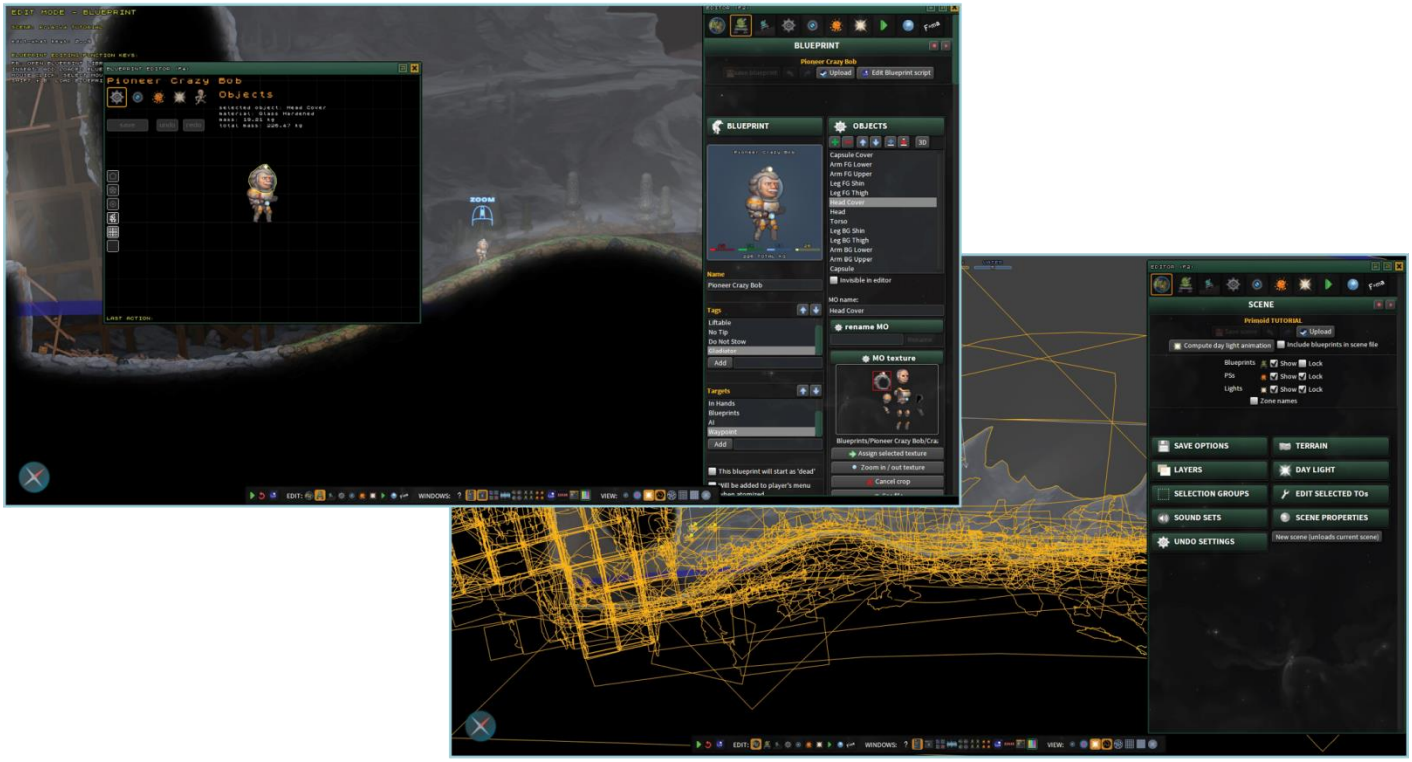
Accessing edit mode

Planetoid Pioneers has two modes: **game mode** and **edit mode**. So far you've (probably) only used game mode where gameplay takes place.

Press the ***F2*** key on your keyboard. What you see next might look similar to the images below (don't worry even if it doesn't; this just means you're in different part of the mode).

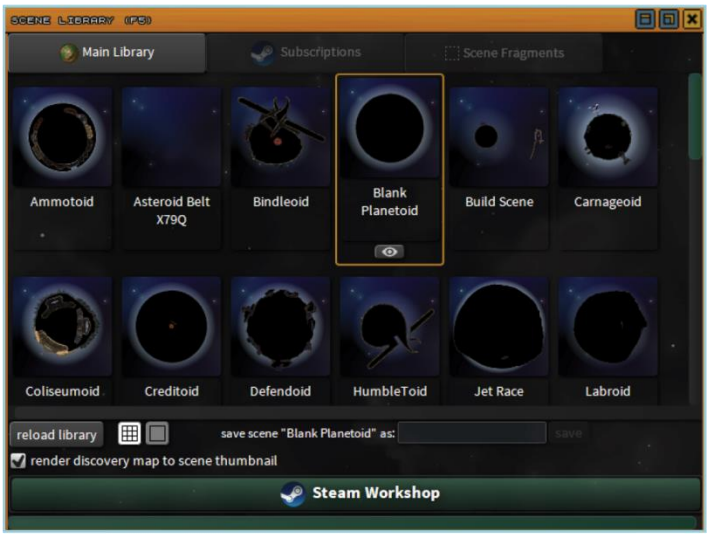
This is how the level looks in edit mode, where everything can be modified and new things can be added. **Edit mode is where the game is created**.

Unlike during gameplay, you can't use a controller here. If you want to go back to playing, just press *F2* again. **You can switch between modes anytime**.

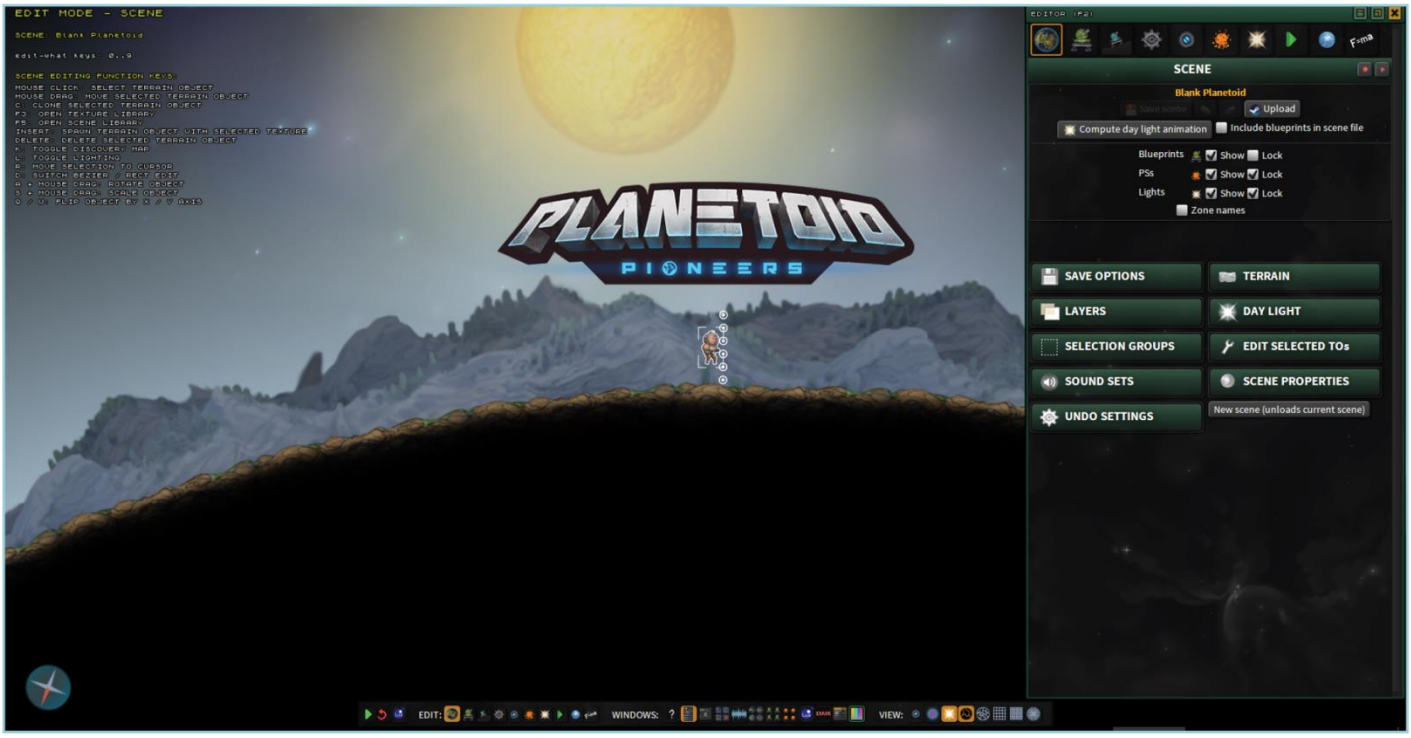


Blank planetoid

The tutorial planetoid is quite complicated, let's start with something simpler. In edit mode press ***F5***. This opens ***Scene library*** window. Here you can see that there are multiple planetoids, and each have their own scene. The one you start in is called *Primoid TUTORIAL*. At the top of the list double click ***Blank Planetoid*** to load it. Once the scene is loaded, **enter edit mode** (***F2***). The view should be a bit less confusing. See the next page for a breakdown of everything you see here.



If there are other windows open besides the one shown on the right of the below image ("*Editor F2*"), you can close them.



What are scenes?

The term *scene* is not unique to this editor, but is commonly used in different game making software. Scenes are smaller, logically related sets of content with specific functionalities (for example a level) that together form a bigger entity (a game). They aren't always levels (or in this case, planetoids). For example, the planetoid map, also known as *Asteroid Belt X79Q*, is a scene that functions as a level selection menu.

Editor overview

First of all, the basic controls are:

» **Move** the view by **right-clicking and dragging** or **pressing space and moving the cursor** or with **arrow keys**

» **Zoom** with **mouse scroll** or **- and + keys**

» As stated previously, no controller controls here

Here you can see the **editor UI broken into smaller pieces**. This is just a brief overview; most of the elements are discussed in more detail later.

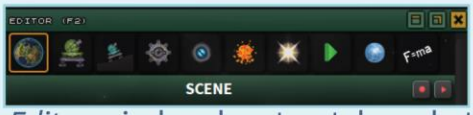
What is UI?

UI stands for *User Interface*. It's not the most relevant term for this guide, but commonly used in computing. It refers to the part of the program that is visible to its user. It's used to interact with the underlying, "invisible" processes. For example, everything you see when you look at the editor is part of an UI. Right-clicking and dragging is interacting with it by telling it to move the view.

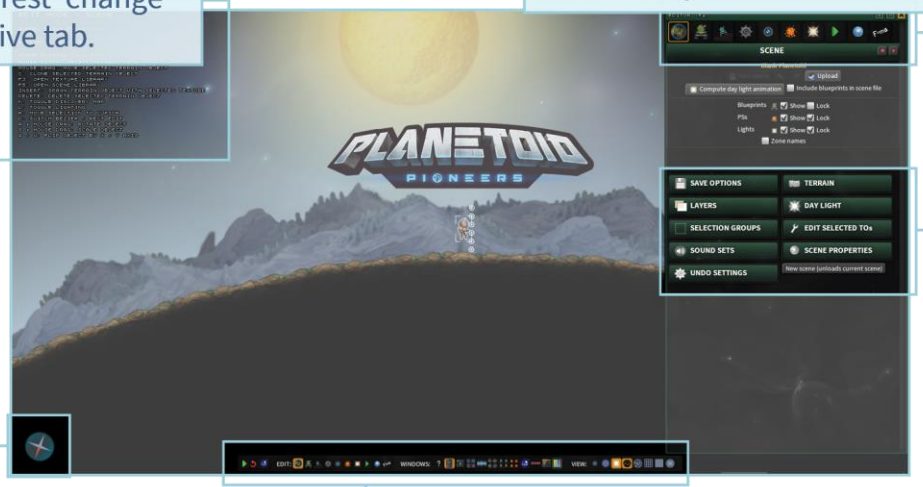
Keyboard shortcuts

On the top left are the **hotkeys** available in the current mode. Keys 0–9 move between tabs in ***Editor* window**, the rest change depending on the active tab.

Editor window



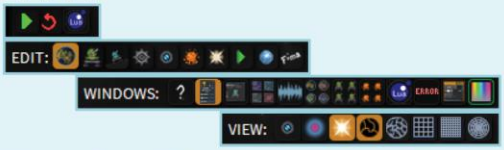
Editor window has ten tabs, what you can do depends on which is selected. They are basically **sub modes within edit mode**. Contents in each tab are grouped under different **panels**.



Compass

Helpful for **navigating the scene**. By default (left image), when moving, the view moves along the surface of the planetoid. Clicking the compass switches to mode (right) where surface shape is ignored. Clicking and dragging the cursor around the compass makes the view rotate around its own centre.

Toolbar



Buttons for different features. The first three are related to the **simulation and debugging**. The ones in group labelled ***Edit*** switch between *Editor* window tabs. ***Windows*** buttons open and close different windows. The ones labelled ***View*** hide and show different aspects of the scene. Most of these buttons have hotkey counterparts.

The game, the editor, & the engine

Game engines are software used for game development; they provide resources and limitations for creating games. An editor is a part of an engine; it's the UI that is used to interact with said engine. Some game companies use engines made by third-party developers while others create their own from scratch. *Planetoid Pioneers* utilises its own *Crush2D* engine.

Whereas game editors and the games made with them are usually separate software, in *Planetoid Pioneers* the editor is part of the game. Since there's so much overlap, here's a breakdown of how the terms are used in this guide: ***Planetoid Pioneers*** refers to the **entire software**, including the game, editor, and engine. **Editor** is **everything visible in edit mode**. **Engine and *Crush2D*** refer to the **underlying logic and technology**.

Helpful tips

If this is your first time using the editor and reading this guide, these probably seem a bit out of context. You can move on for now, **just remember that this page exists** and come back once you think these might be useful.

Abbreviations used in *Planetoid Pioneers* and this guide
BP → BluePrint; MO → Movable Object; PS → Particle System; TO → Terrain Object

Browsing libraries faster
If you know the name of the asset you are looking for, you can find it faster by typing its initial letter on the keyboard while a library window is open and active.

Comprehensive edit mode hotkey list
In *Help* window (*Esc*, *F1,* or the toolbar's *?* icon) you can find a comprehensive list of edit mode hotkeys (all of them won't be discussed in this guide).

Context-sensitive help
This guide doesn't go through everything the editor has to offer. If there's something you don't know the purpose of, you can try hovering the cursor over it to make a tooltip appear.

Controlling the character while in edit mode
Press *Ctrl* + *F2* to make the character controllable in edit mode, toggle the control off with the same key combination.

Dot as decimal separator
When inputting numeric values, note that the editor uses dot (.) as decimal separator.

More *Planetoid Pioneers* tutorials
In *Help* window (*Esc*, *F1,* or the toolbar's *?* icon) there's a *Guides* button that links to *Steam*'s *Guides* page where you can find more *Planetoid Pioneers* tutorials.

Opening and closing all the panels in an *Editor* window tab
At the top of *Editor* window you can see the name of the currently active tab. Click that to open or close all the panels.

Separate windows
You can open some of the editor's internal windows as entirely separate ones by clicking the minimise icon on top right (two icons left of *X*). Bring them back by clicking minimise again.

Sharing content
It is possible to share the content you make with others. Check the section *Sharing your content* at the end of this guide for more info.

You should now have a very basic understanding of the editor's user interface. The next step is knowing how to use it. The best place to start is creating a base for the game world, your own planetoid. This section focuses on Scene tab.

Customisable planetoid

Before doing anything irreversible to the existing assets, make your **own copy of the** *Blank Planetoid* **scene**.

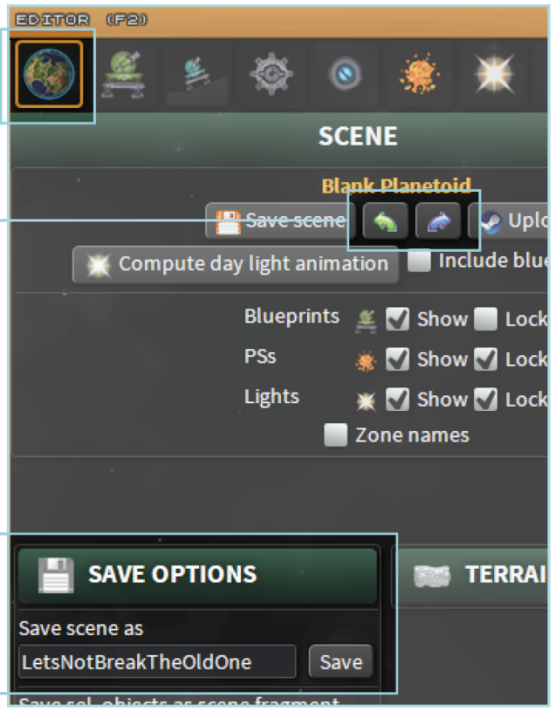
Choose ***Scene* tab** in ***Editor* window** (the first one from left, alternatively press *1* or click the first icon after *EDIT* in the toolbar). There you can see a panel titled ***Save options***. Click it to expand it. Under ***Save scene as*** type a **unique name** for the scene (if you choose one that is already taken, the existing scene will be overridden). Next, click ***Save***, and ta-dah, you're now in your own scene. You can check the name of the active scene on the top left corner of the editor. **From now on you can save the changes you make with the *Save scene* button**.

But what if you already made changes you aren't proud of and don't want saved? Luckily the editor has a quite extensive **undo** feature. Either use the key combination ***Ctrl* + *Z*** or the **green arrow** button below the tab name. **Redo** works with ***Ctrl* + *Shift* + *Z*** or the **blue arrow** button next to the green one. Alternatively you can reload the scene through *Scene library*, but everything unsaved will be lost.

The same **undo/redo feature is available in other parts of edit mode as well**.

Assets

Asset is a blanket term for all the building blocks of a game available in the editor. Assets include scenes, images (textures), sounds, music, characters, scripts, etc.



Constructing a planetoid

Now it's time to start making your own planetoid. Staying in ***Scene* tab**, press ***K*** and ***L***. The former toggles discovery map (see the next page for more info), the latter lighting. Without them the terrain is easier to see. Alternatively, in the bottom toolbar, switch off the third and fourth icon right of *View*.

The planetoid consists of **terrain objects**, or **TOs**. During gameplay TOs are **static and can't be destroyed**.

Remember that **you have to be in *Scene* tab to modify the terrain**. Also note that the playable character, **pioneer, is not a TO**, and trying to modify it as such does not work as you might expect. It's best to leave Bob be for now.

You can **select** one TO by **clicking**, or multiple by **clicking and dragging** a rectangle around them. You can **add** more TOs to your selection by holding ***Shift*** and **remove** from it by holding ***Ctrl***. They can be **moved** by **dragging, deleted** with ***Delete*** and **copied** with ***C***.

Once you have selected a part of the terrain, the options you have for **modifying** it become visible. Hovering over the buttons shows what each does and the matching hotkeys.

Besides just copying, you can **create new TOs** from textures. Press ***F3*** (or toolbar, third icon after *Windows*) to open ***Texture library***. It contains all the existing textures. The ones used here are under *Terrain/Dirt Brown Moss*. However, you can create a TO from any texture by **dragging it to the scene**. A collider (see the next page) for it is automatically created.

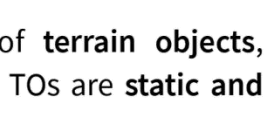
If that's not enough, you can also add, or **import, your own textures**. Just **drag a PNG file** into the scene and it gets added to the library, under ***!-import-*** folder. Dragging it on top of the library window adds it to the currently open folder. Note that the images should have **transparent backgrounds**.

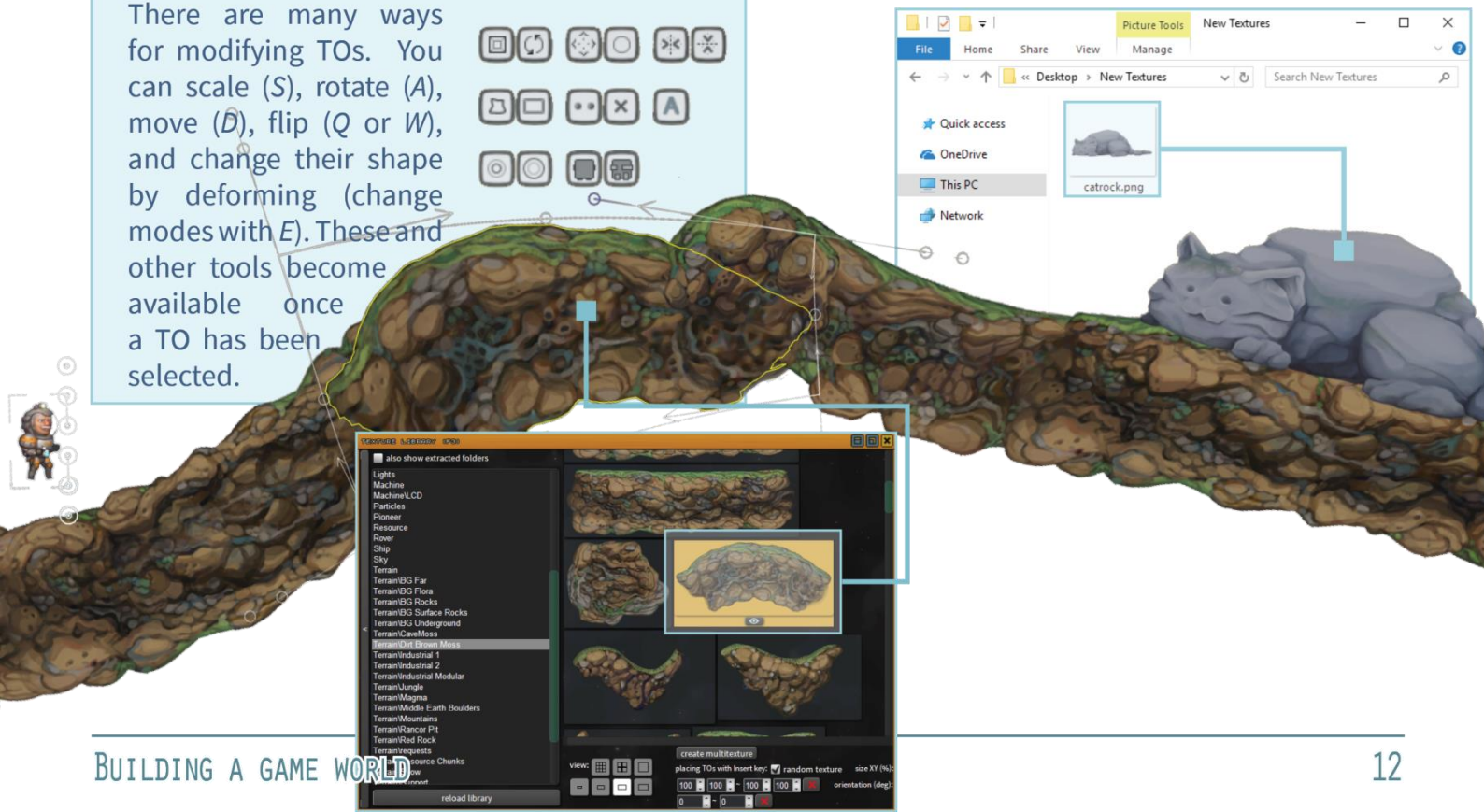
Knowing these basics should be enough to get you started. On the next pages there are some more advanced tips.

Remember to click the *Save scene* button before quitting!

Modifying options

There are many ways for modifying TOs. You can scale (*S*), rotate (*A*), move (*D*), flip (*Q* or *W*), and change their shape by deforming (change modes with *E*). These and other tools become available once a TO has been selected.





Discovery map

Discovery map hides the areas the player hasn't seen yet. It also fixes "flaws" in map design since it hides areas that can't be accessed. This means that you don't have to fill in everything with TOs. Clicking *Compute day light animation* on top of *Editor* window updates it to match the terrain's shape.

Colliders

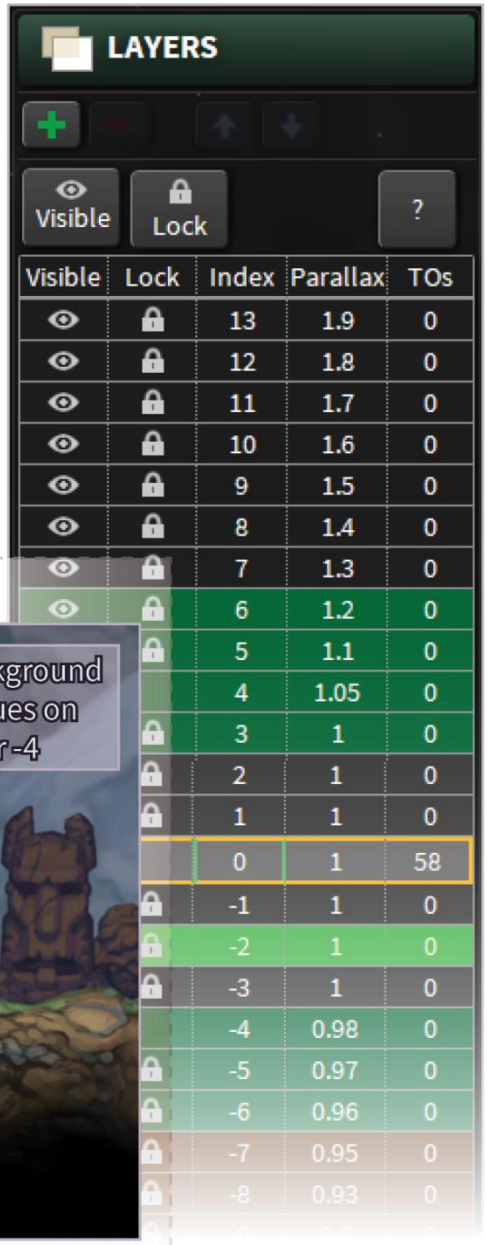
Colliders, or hitboxes, are edges of objects that the engine uses to detect collisions between two (or more) of them. In other words, they are used to check if objects are touching each other. They are invisible during gameplay but can be seen as yellow lines in edit mode.

Advanced terrain creation

The game world consists of several **layers** stacked on top of each other. Layers can be used to give the scene more depth or for guidelines for editing.

They are listed in ***Layers* panel**, the smaller the index, the further back the layer. Most of these are for visuals, only layer 0 is reserved for gameplay. Everything interactable, also the ground, should be placed there.

Move between layers by clicking the index numbers. Layers -3 to 3 detect collisions. **Eye icons** toggle layers' **visibility** in edit mode. **Locks** affect **whether or not they can be modified**. **Colours and notes** don't change anything but are helpful for **organising** them.



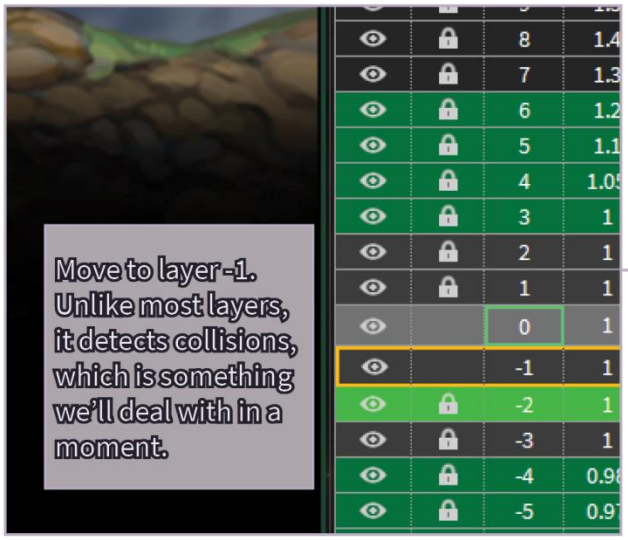
Foreground and background

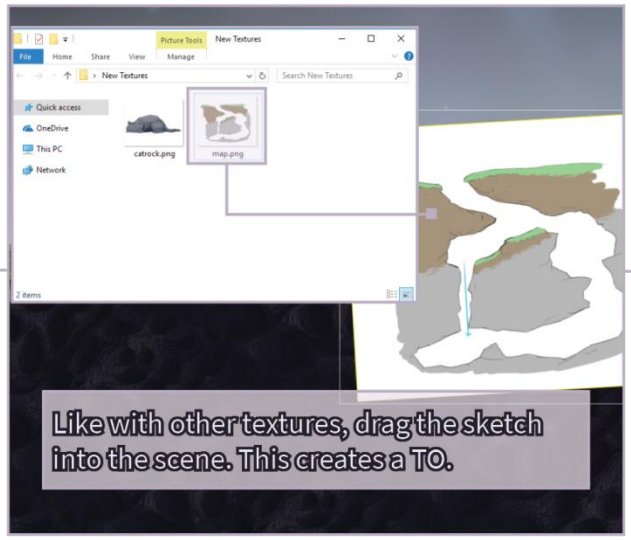
These don't affect the gameplay but add more depth to the world. Background TOs are placed on layers further back, foreground ones in front. Layers -3 to 3 detect collisions, so these TOs should be placed on layers -4 and below and 4 and above.

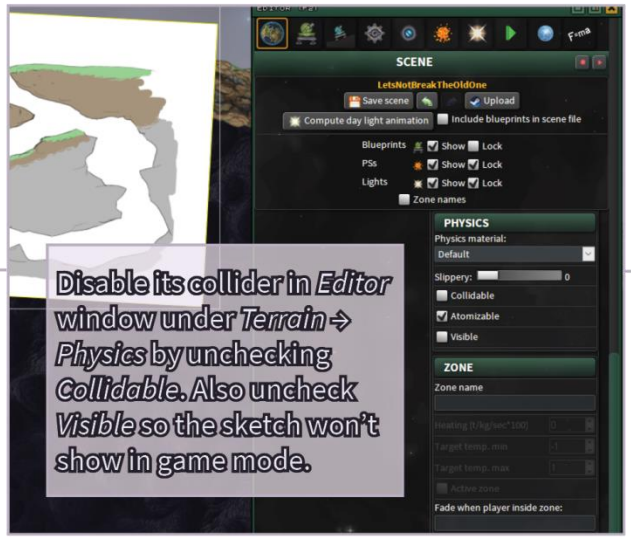
Hint: among the modifying options there are tools for sending a TO a layer backward (*Page Up* key) and bringing it forward (*Page Down*). The selected TO's layer is also shown. *Bring to front* (*End*) and *Send to back* (*Home*) don't change the layer but the order of TOs within a layer.

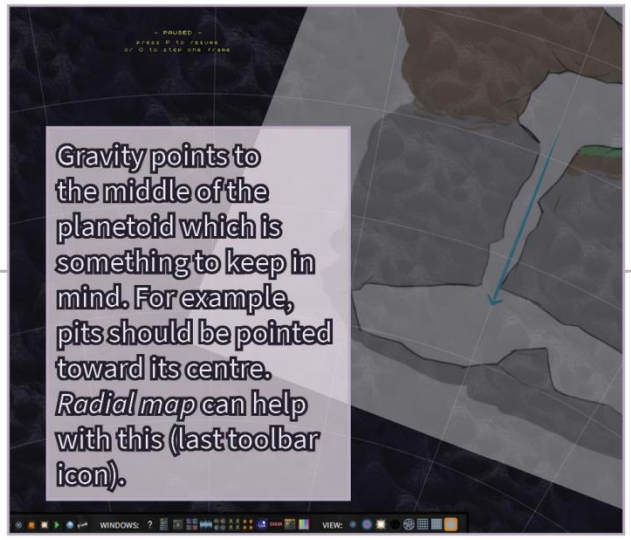
Terrain based on image

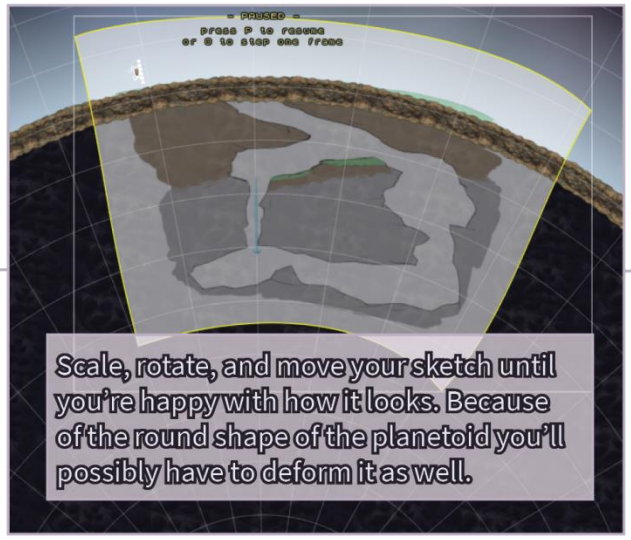
It's possible to use an image as a reference for a planetoid. This means that you can sketch a level outside *Planetoid Pioneers*, import it to the editor, and then build your planetoid on top of it. Since it won't be in the final level, it doesn't have to have a transparent background.

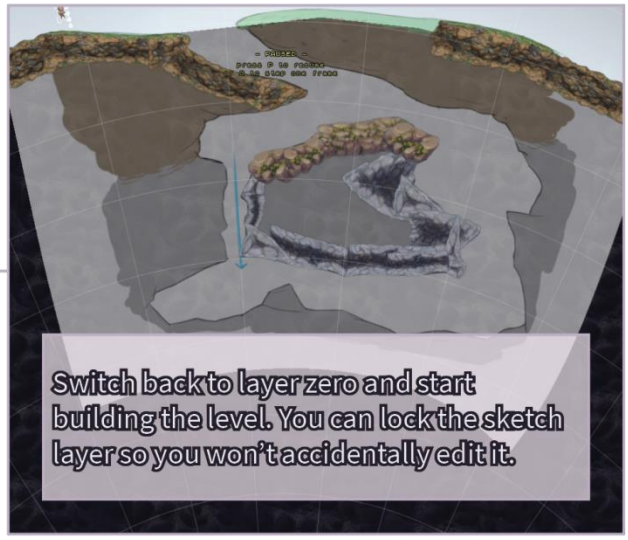






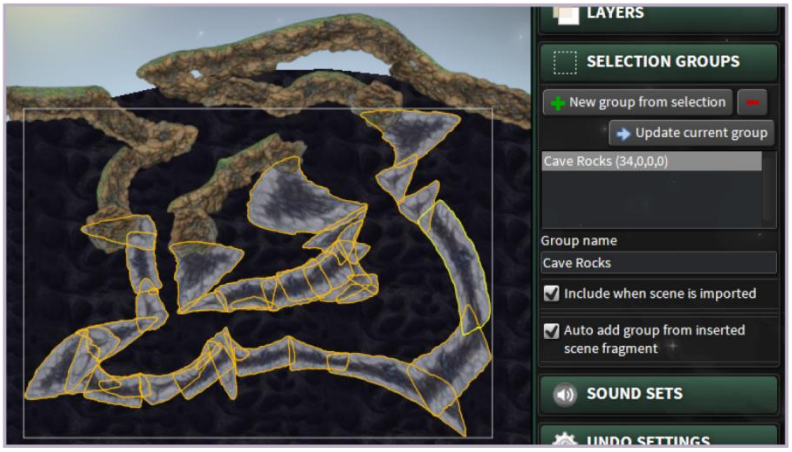






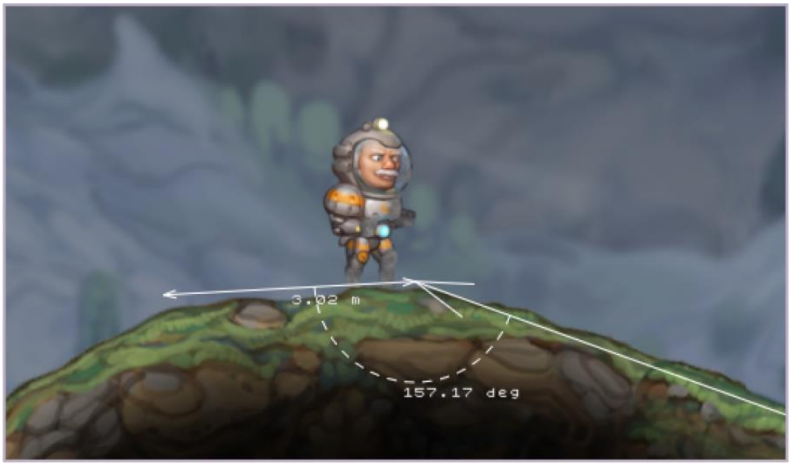
Selection groups

Selection groups are things that can be surprisingly helpful at times. In *Selection groups* panel you can save the currently selected TOs as a group that can be easily selected again afterwards. Note that if you click *Update current group* without first selecting said group, it will be overridden with the current selection instead of adding to the previous one.



Gauges

One quite well hidden but possibly useful feature is the ability to create guidelines that show the distance between two points. Press the *Tab* key and click and drag. If you draw two consecutive lines right next to each other, you'll also get the angle between them. Remove all lines with *Ctrl* + *Tab*. They can be created in both, game and edit modes.



Troubleshooting

Can't move the character/nothing happens in game mode

» Check that you're in fact in game mode (move between modes with *F2*).
» Press *P*; the game might be paused.
» Try resetting the scene with *F9.*
» Go back to edit mode with *F2*, reset the scene there with *F9*, and then go to game mode with *F2.*

Can't see modifying options when a TO is selected

» Press *F*; this toggles their visibility.
» Move the view; they might be hidden behind a window.

Can't select TOs

» Check that you're in edit mode (move between modes with *F2*).
» Check that you're in *Scene* tab (press *1* or select the first tab in *Editor* window).
» Check that their layer is unlocked (see page 13).

Don't know the layer of a TO

» Select the TO, if you can do this, its layer is visible below the modifying options.
» Otherwise start toggling the visibility of layers by clicking the eye icons in *Layers* panel; once the TO disappears, you've found its layer.

The character starts at a wrong place/inside the ground in game mode

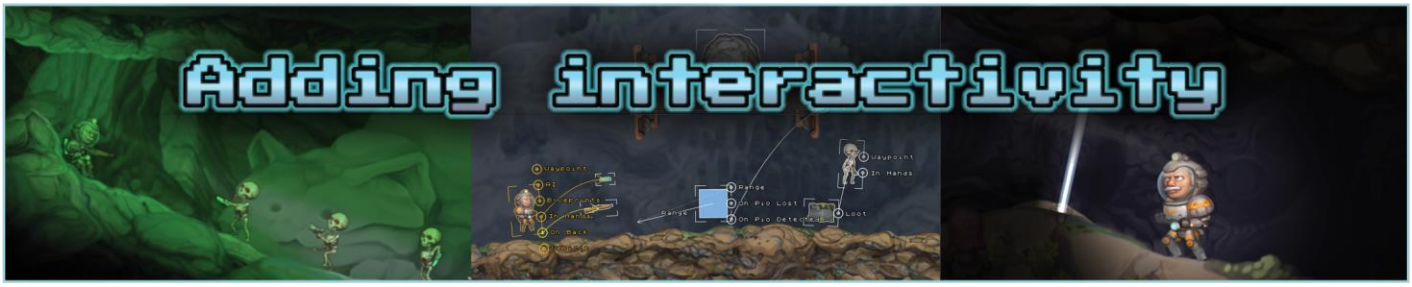
» Press *F9* to reset the scene.
» Check the next section (*Adding interactivity*, page 18) for instructions on how to permanently change the starting position.

The selected TO deforms instead of moving when dragging

» If the TO is small, the editor might interpret trying to move it as an attempt to change its shape; avoid this by pressing *D* when dragging.

TOs get placed on a wrong layer

» Check that you have the correct layer selected in *Layers* panel; TOs can be placed even on locked layers.
» To change their layer, select the TOs you want to move and press *Page Up* key (send one layer backward) or *Page Down* key (bring one layer forward) until they're on the correct layer.



You've created a base for your level. However, everything is static and you can't do much besides run around. It's time to start adding interactable objects to the game. In this section you'll be introduced to movable objects, concept of blueprints and Activity tab.

MOs, blueprints, and assemblies

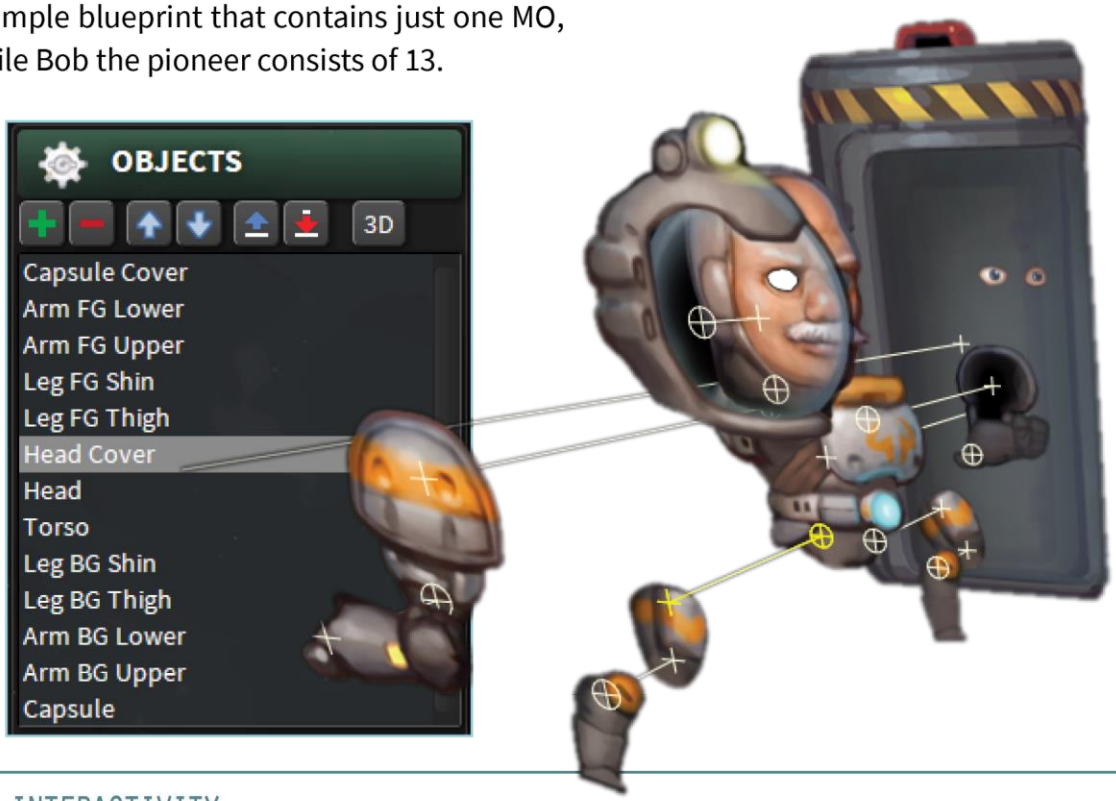
The worlds created in *Planetoid Pioneers* consist of two types of objects. The first group, TOs, we went through in the previous section. The second, more interesting type are **Movable Objects** or **MOs**.

MOs are **objects affected by the physics**. Like TOs, MOs can be created from any texture. However, an MO on its own has very little functionality. Instead, they should be **attached to a blueprint**.

Blueprints, or **BPs**, are **combinations of different components** and can contain one or more MOs. For example, a wooden box is a simple blueprint that contains just one MO, while Bob the pioneer consists of 13.

Everything you can interact with during gameplay, including characters, enemies, weapons, boxes, doors etc. **is an instance of a blueprint**, also known as an **assembly**.

We'll come back to modifying blueprints in a later section and will for now be using existing ones. However, to help you understand how multiple MOs can be combined to create a more complex entity, below is a list of the MOs used to create Bob and a 3D view of the seemingly 2D character that shows how these pieces are connected.



Activity tab & adding assemblies

Move to ***Activity* tab** in ***Editor* window** (eighth icon from left, alternatively press *8* or click the eighth icon after *Edit* in the toolbar). You can also quickly **toggle** between *Scene* and *Activity* tabs by clicking the **mouse wheel**.

***Activity* tab handles the "game" part of the scene**. It holds all the interactables, the instances of blueprints present in the scene. They are listed under *Activity blueprints* panel. Assuming you haven't added new ones, *Standard Pioneer* (Bob) is the only one.

Activity script can, for example, be used to define the win and lose conditions of the scene. However, this requires programming, which is something we won't go into in this guide so leave that as it is.

The available blueprints can be found in the ***Blueprint library*** (not to be confused with the in-game blueprint library), open it with ***F6*** (or 7th icon after *Windows* in the toolbar). You can **drag any blueprint into the scene to create an assembly**. Some of them (like lamps and drawbridges) can be attached to the terrain, the plus signs signify the spots for this.

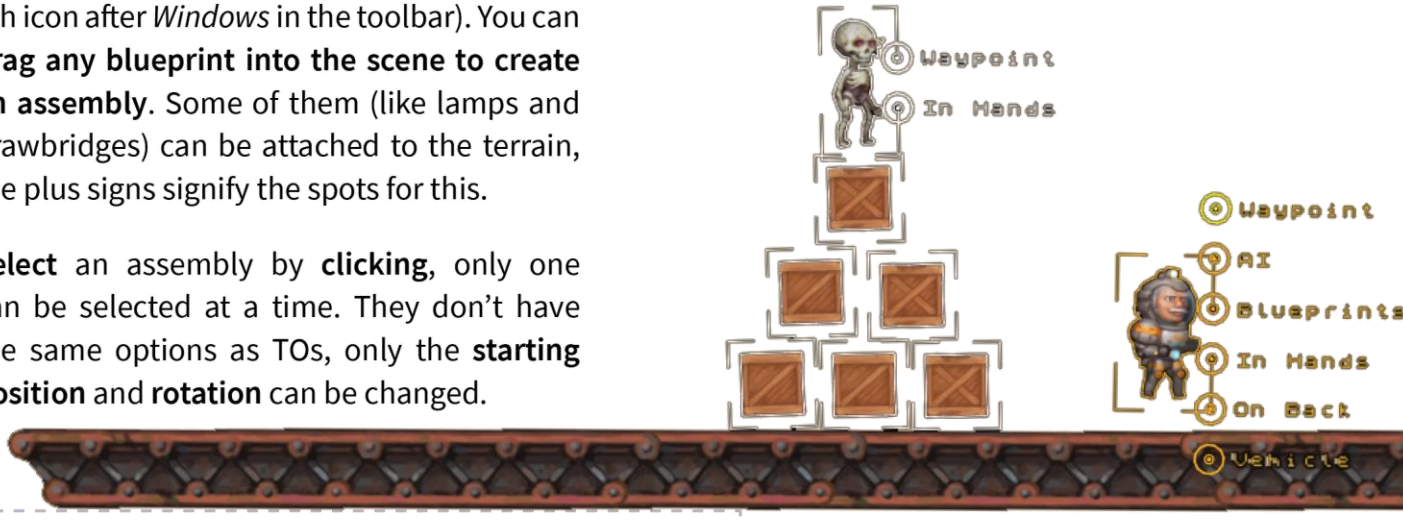
Select an assembly by **clicking**, only one can be selected at a time. They don't have the same options as TOs, only the **starting position** and **rotation** can be changed.

Change the **position** simply by selecting and **dragging**. **Rotation** is changed in *Editor* window under ***Selected blueprint* panel** by changing the value of ***Orientation***. Like TOs, assemblies can be **removed** with ***Delete***.

The **undo/redo** feature works the same way as it did in *Scene* tab. **Save** changes by clicking ***Save scene*** at the top of the window.

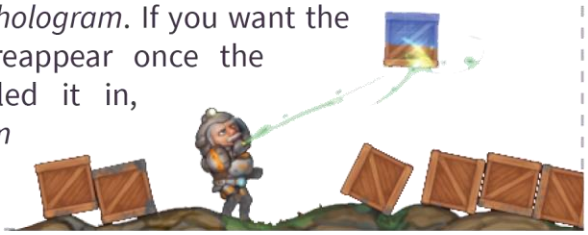
Preview/restart the scene with ***F9***, move to game mode with ***F2***. *F9* starts the **physics simulation**, so assemblies that are affected by gravity fall to the ground. The ones that detect collisions and are stacked together "explode" as the engine doesn't allow colliders inside each other.

Don't forget to **save your changes** once you're done.



Holograms

When you were playing the tutorial level, you may have run into blue, glowing *holograms*. They are bases where the player can fill in assemblies even if they don't have the blueprints in their (in-game) blueprint library yet. If you want an assembly to start off as a hologram, in *Editor* window under *Activity* tab's *Selected blueprint* panel check *Start as hologram*. If you want the hologram to reappear once the player has filled it in, check *Hologram auto recreate*.



About blueprints

The idea of blueprints was not invented for this editor. The term varies, but there are other engines that offer a similar possibility to create templates that can be used several times in different scenes.

Creating connections

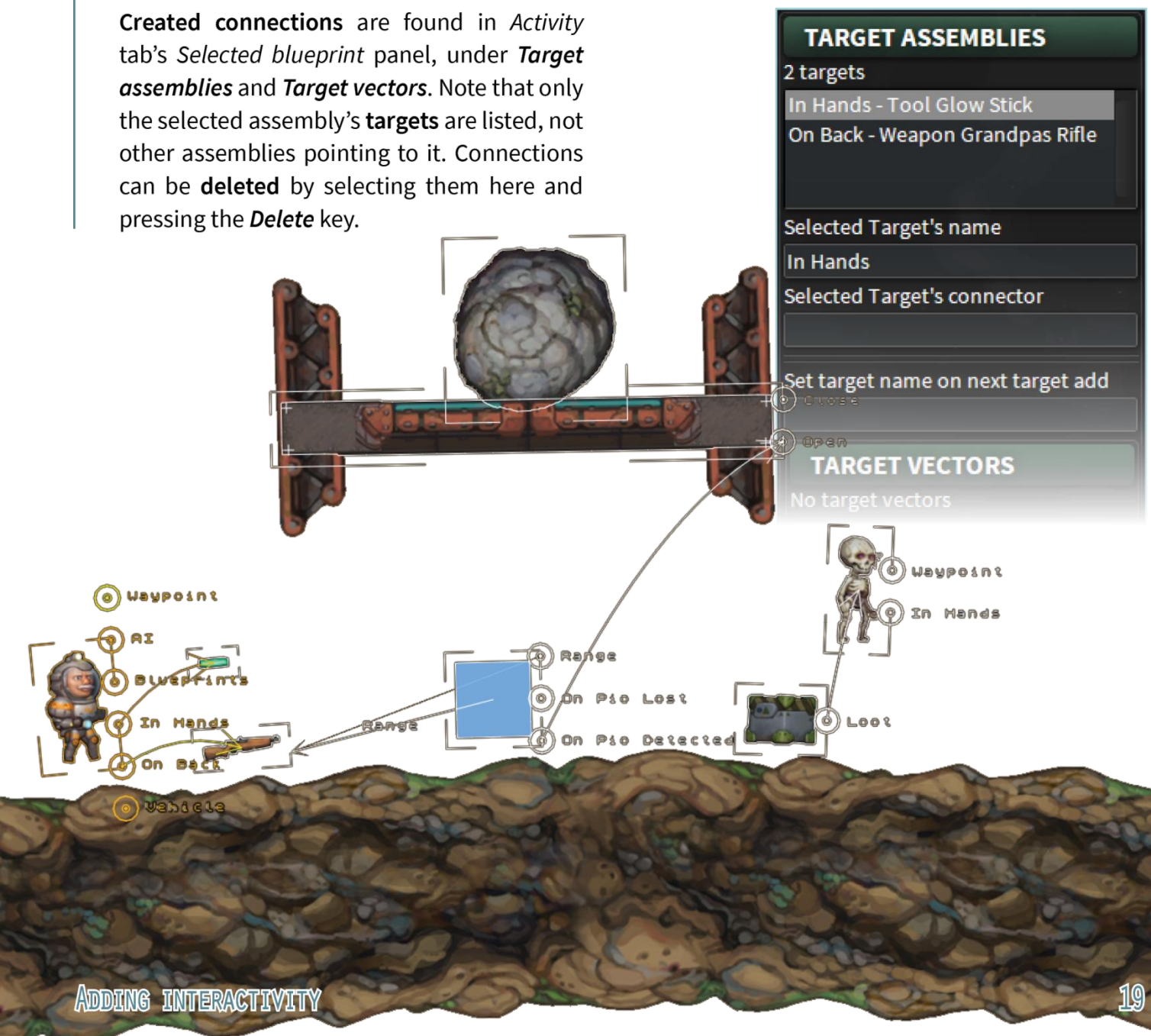
The circles next to some of the assemblies are **connectors** for creating different kinds of connections. These can be straightforward, like **linking assemblies together** (holding an item, placing loot in a chest), or more complex, like making a change in one assembly **trigger** a **reaction** in another (breaking a cable to open a locked door).

Connections are created by selecting an assembly with connectors and **dragging an arrow** from one of them to its desired **target**. Targets can be other **assemblies** (assemblies themselves or their connectors) or **vectors** (spots in the map or even "nothing"). We'll go through examples for each of these.

Created connections are found in *Activity* tab's *Selected blueprint* panel, under ***Target assemblies*** and ***Target vectors***. Note that only the selected assembly's **targets** are listed, not other assemblies pointing to it. Connections can be **deleted** by selecting them here and pressing the ***Delete*** key.

Connectors have been individually defined for existing blueprints which means that there's a lot of variation between them; **what you can do with one blueprint you can't necessarily do with another**. Keep this in mind when designing interactions. Also note that there is usually a **specific way how each connector should be used** and that the editor **doesn't stop you from making invalid connections**.

Since there are so many blueprints with different connectors, we won't be going through all of them. Instead this guide lists some of the most basic ones and gives a few examples how they can be used.



Simple connections

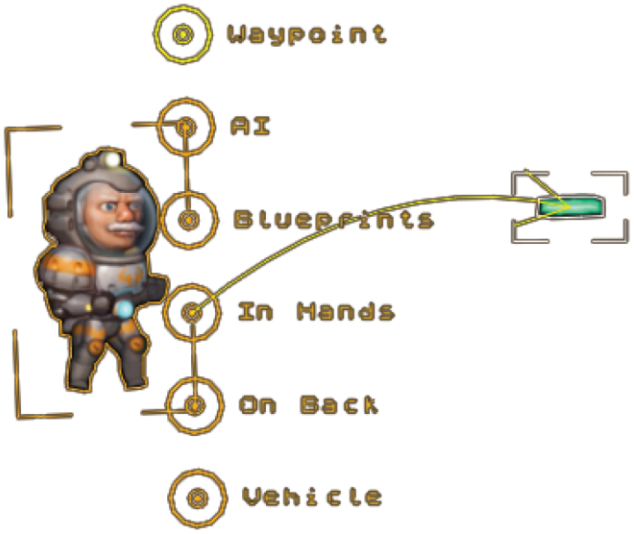
First the most basic connection: linking assemblies together when a scene is started. For this we'll be looking at four of Bob's connectors in more detail (*Blueprints*, *In Hands*, *On Back*, *Vehicle)*.

Maybe Bob should start off with a glow stick in his hand. In ***Blueprint library*** (*F6*), find ***Tool Glow Stick*** and drag it into the scene somewhere near Bob. Next, **select Bob** and drag an arrow from ***In Hands*** to the **glow stick**. Preview/restart the scene with ***F9***, ***F2*** to move to game mode (***L*** to toggle lights to see the effect).

The other three connectors work the same way: find a blueprint, drag it to the scene and drag an arrow to it.

On Back **places an item on Bob's back,** ***Vehicle*** **places Bob in a vehicle** (look for rovers in *Blueprint library*). ***Blueprints*** starts Bob off with the chosen blueprints **already in his in-game blueprint library**.

Pioneers aren't the only ones that can hold items. *Fauna Zombies* also have the same *In Hands* connector. So if you want to give your enemies weapons (or glow sticks), that is possible. Other places where these connections can be used are, for example, chests with specific loot.



In the table below, there are examples of connectors that can be used for these simple connections (ordered alphabetically by connector name). To reiterate: arrows are dragged **from** these **connectors to** target **assemblies**.

Connector	Blueprints	Description
Blueprint	Loot Blueprint	Blueprint player finds in the chest
Blueprints	Pioneers	Blueprints a pioneer has in its library when the game starts
Create Cable	Trigger Cable	Connecting this to another assembly creates a cable between them
Create To	Flora Liana Structure Rope	Connecting this to another assembly creates a rope between them
In Hands	Fauna Zombie Junk Skeleton Pioneers	Assembly a pioneer/skeleton is holding when the game starts
Loot	Loot Blueprint Chest	Loot the player finds in the chest
On Back	Pioneers	Assembly a pioneer has on its back when the game starts
Pair	Structure Teleporter	Pairs teleporters together
Vehicle	Pioneers	Vehicle attached to a pioneer at the start of the scene

Triggers and reactions

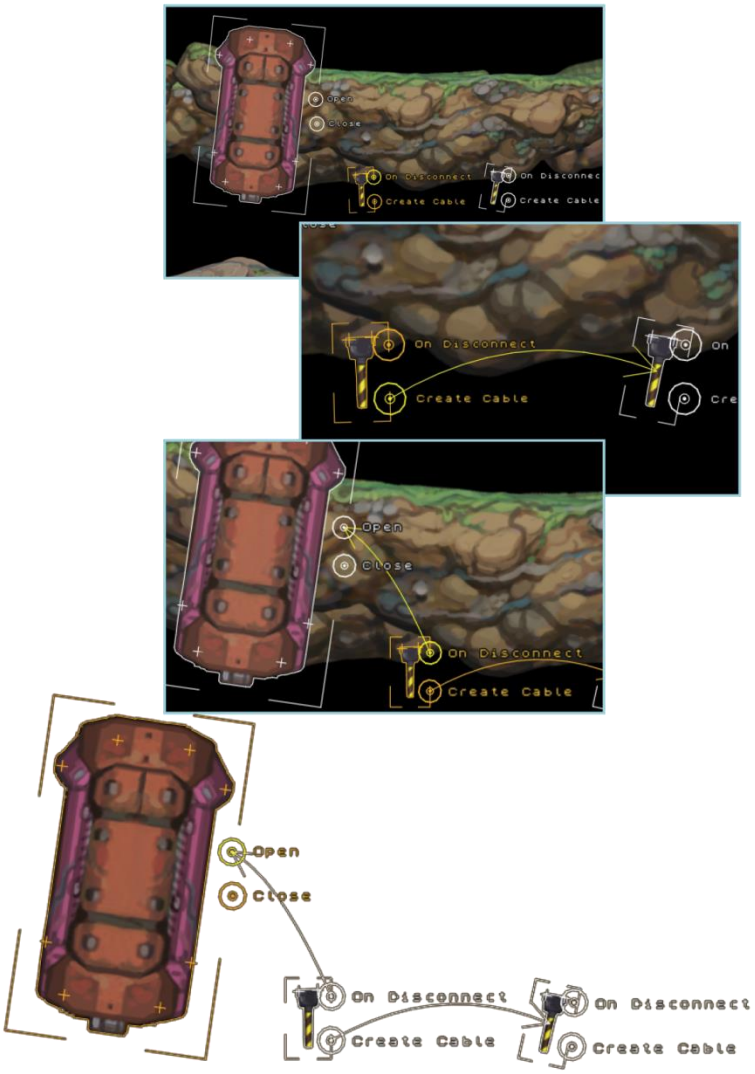
In the tutorial level there were locked doors that could be opened by breaking a cable next to them. They are a good introduction to how triggers and reactions work.

Open ***Blueprint library*** (*F6*), look for ***Structure Covered Door*** and place it into the scene. Next, look for ***Trigger Cable*** and drag **two** of them in.

Select one cable and drag an arrow **from *Create Cable* connector to** the other cable **assembly**. As the name implies, this creates a cable between these two points.

Finally, select a cable and drag an arrow **from *On Disconnect*** (trigger connector) **to** the door's ***Open*** (reaction connector). Now when the cable breaks the door reacts by opening.

Of course, there are a lot of other possibilities as well. Below is a short list of some of the available **triggers** (arrows are dragged **from** these **connectors**...)



Connector	Blueprints	Description
After Delay	Reactor Trigger Delayed	Triggers a reaction in connected assembly after *Delay*
Is Down/Is Up	Structure Switch Sandstone	Triggers a reaction in connected assembly when this is down/up
On Activated Trigger Once	Reactor Trigger Once	Triggers a reaction in connected assembly, works only once
On Disconnect	Flora Liana Trigger Cable	Triggers a reaction in connected assembly when cable is broke
On Pio Detected/ On Pio Lost	Trigger Proximity Invisible	Triggers a reaction in connected assembly when pioneer is within/has left the *Range*
On Time Up	Structure LCD Countdown Timer	Triggers a reaction in connected assembly when the *Time Limit* is reached

Note that some of these are closely tied to target vectors, which we'll go through next.

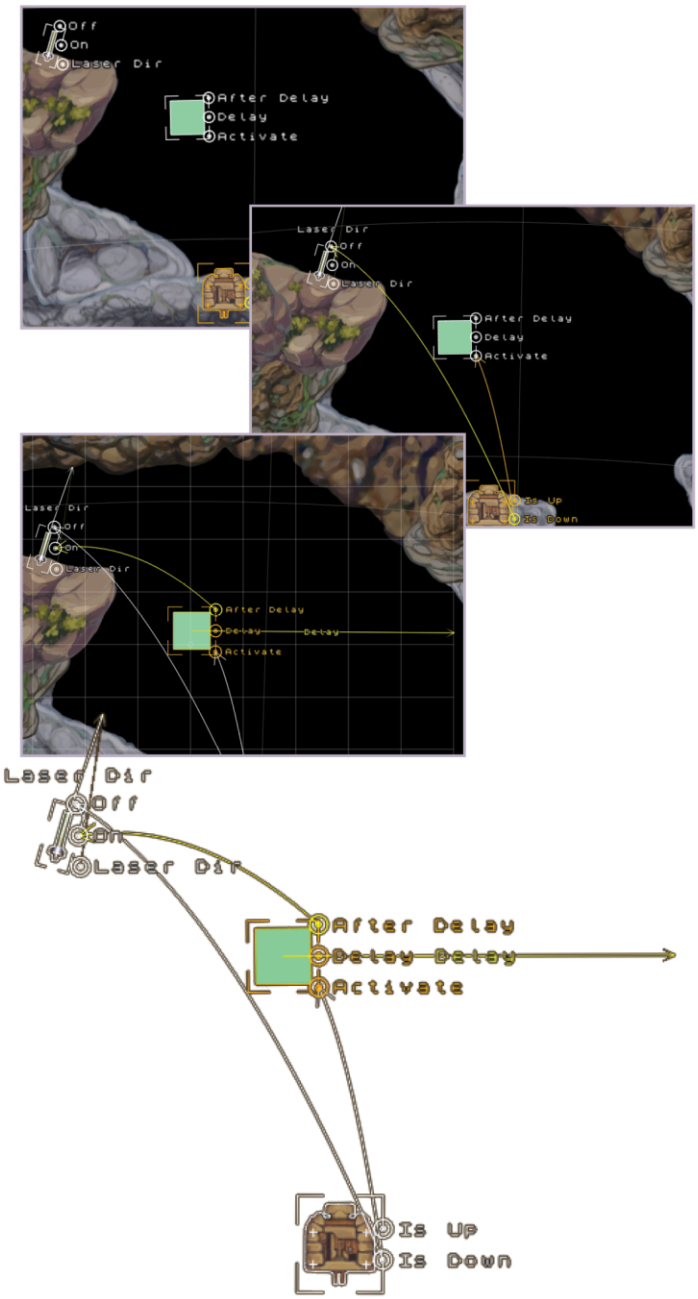
And here's one of **reactions** (...**to** these **connectors**).

Connector	Blueprints	Description
Activate	Reactors	Activates this assembly
Close/Open	Structure doors Structure Latch Structure trap doors	Closes/Opens this door
Discover Now	Lamps	Clears the discovery map around this lamp even if the player hasn't been there yet
Off/On	Lamps Machine Laser Emitter	Turns off/on this assembly
Sensors	Structure Moving Platform	Brings the platform to the *Stop* closest to this assembly
Turn Off/Turn On	Lamps Structure Moving Platform Structure Teleporter Structure trap doors Traps	Turns off/on this assembly

These tables are by no means comprehensive, there are a lot of other blueprints and connectors available. These only serve to give an overall idea of what can be done with them.

Using vectors as targets

So far the targets have been other assemblies or their connectors. Next, let's look at **target vectors**. They can point to specific points on the map, define an area around an assembly or even be used as a way to input a numeric value.



Let's make a bit more interesting "door". Drag in ***Structure Switch Sandstone***, ***Machine Laser Emitter***, and ***Reactor Trigger Delayed***. Place the laser where you'd want a door to be and the switch a bit further away from it. The reactor won't be visible during gameplay, but for clarity's sake place it somewhere between the previous two.

Drag an arrow **from** the laser's ***Laser Dir*** **to where you want it to point**. Drag arrows **from** the switch's ***Is Down*** **to** the laser's ***Off*** and the reactor's ***Activate***. Now when you press it, it will turn off the laser and activate the reactor.

Drag an arrow **from** the reactor's ***Delay***, only its **length** matters. Press ***G*** to bring up a **grid**. One unit is 2x2 meters, so for a delay of about ten seconds draw a 5-unit-long arrow. Finally, drag an arrow **from** the reactor's ***After Delay*** **to** laser's ***On***. Now the reactor turns the laser back on after ten seconds have passed.

Here are some other examples of connectors that are used for **target vectors**. Drag an arrow **from** these **to** somewhere in the **game world**.

Connector	Blueprints	Description
Delay	Reactor Trigger Delayed	The length of the target vector is used as a numeric value for delay in seconds
Laser Dir	Machine Laser Emitter	The direction of the target vector is used as the direction of the laser (length doesn't matter)
Range	Trigger Proximity Invisible	The length of the target vector is used as numeric value for range in meters
Stops	Structure Moving Platform	Marks the spots between which the platform should move, can be multiple
Time Limit	Structure LCD Countdown Timer	The length of the target vector is used as a numeric value for the time limit
Waypoint	Bots Fauna Silverback Fauna Zombie Pioneer Crazy Bob	Spot on the map this assembly should move towards. Note: enemies prioritise attacking a close by player over waypoints. Also: if the pioneer is controlled by a player, this does nothing.

Troubleshooting

Accidentally double-clicked a blueprint in *Blueprint library*, a new window opened and the active *Editor* window tab changed
» No harm done, just close the unwanted window and move back to *Activity* tab.

Assemblies aren't listed under *Activity blueprints* panel
» Reset/preview the scene with *F9.*
» Make sure that there are assemblies in the scene and that you didn't accidentally add them in a tab other than *Activity*.

Assemblies fall through the ground
» Place them so that the plus signs are on top of the terrain.
» Try resetting the scene (*F9*).

Assembly moves when trying to drag an arrow
» Select the assembly first (outline turns yellow), then start dragging connections.
» Make sure you're dragging from connector circles, not outside them.

Can't see connectors
» Check that you're in *Activity* tab.
» Not all assemblies have connectors.

Can't select assemblies
» Check that you're in *Activity* tab, temporary assemblies can be dragged in in other tabs as well.

Connection doesn't do anything
» Check that the connection moves the right way (for example, from trigger to reaction).
» Check that the arrow goes to the correct connector or, if it links two assemblies together, to the target assembly instead of a connector.
» The editor doesn't stop you from making bad connections. Make sure that the connection is valid.

Made an unintended connection
» Use *Ctrl* + *Z* or the green arrow to undo.
» Select the unwanted connection in *Target assemblies*/*Target vectors* and press *Delete*.

Nothing happens when *F9* is pressed/can't move the character
» Press *P*, the simulation might be paused.
» Check that you're in game mode, simulation can be on even in edit mode.

Scene doesn't restart/everything is broken when restarting
» Press *F9* again.
» Go to game mode (*F2*) and try restarting there (might still take a couple tries).

Toggling between modes with mouse scroll doesn't work
» This seems to happen if the scroll is pressed in *Scene* tab when the cursor is on top of an assembly's starting spot; move the mouse and try again.
» Use alternative options (*Editor* window tabs, toolbar icons, *1* and *8* keys).

Now that you know how blueprints are used, it's time to learn how to customise them. In this section you'll be introduced to Blueprint tab and editor, making changes to individual MOs, and joints.

Editing assemblies vs. editing blueprints

In this section you'll be introduced to altogether four tabs: ***Blueprint***, ***Assembly***, ***Object***, and ***Joint***. Before delving into the main subject, let's quickly go through the difference between making changes to assemblies and blueprints.

Both assemblies and blueprints can be edited; the difference is that **changes made to assemblies can't be saved while those made to blueprints can**.

Editing assemblies gives and opportunity to **test** what different values in different places do. It's possible to move to game mode (*F2*) with them intact, but they'll be **restored back to original ones once the scene is reset** (*F9*).

How these changes, both permanent and temporary, are made will be discussed on the following pages.

Reminder

Blueprints are combinations of different pieces (including MOs). Multiple copies of them can be brought into scenes, these instances of blueprints are called assemblies.

Assembly tab

***Assembly* tab** (third from left, *3*, or 3rd icon after *Edit* in the toolbar) is the least interactive one of the tabs discussed here. It shows **information related to the currently selected assembly**. The information includes target/targeting assemblies, MOs, joints, etc.

Changes, permanent or temporary, **can't be made** here, but it's helpful for checking info of a specific assembly.

Blueprint tab & Blueprint editor

***Blueprint* tab** (2nd from left, *2*, 2nd icon after *Edit* in the toolbar) **manages a blueprint's components**. It's a bit similar to *Assembly* tab as it compiles information related to a blueprint, difference being that here **changes can be made**.

Although the management happens here, some of the **more complex parts have their own places for more precise adjustments**. For example MOs, joints, particles, lights, and poses can be added, but more thorough modifications are done elsewhere.

The currently **active blueprint is the one you've loaded to the editor,** i.e. the one you can see in *Blueprint editor* window.

***Blueprint editor* window** (*F4* or 3rd icon after *Windows* in the toolbar) offers a **visual representation** of the blueprint currently being edited and an opportunity to make **changes in a more intuitive way** than just modifying values in *Editor* window.

Blueprint editor window has **different modes** for editing **movable objects**, **joints**, **particles**, **lights**, and **poses**. The icons on top left are for moving between them.

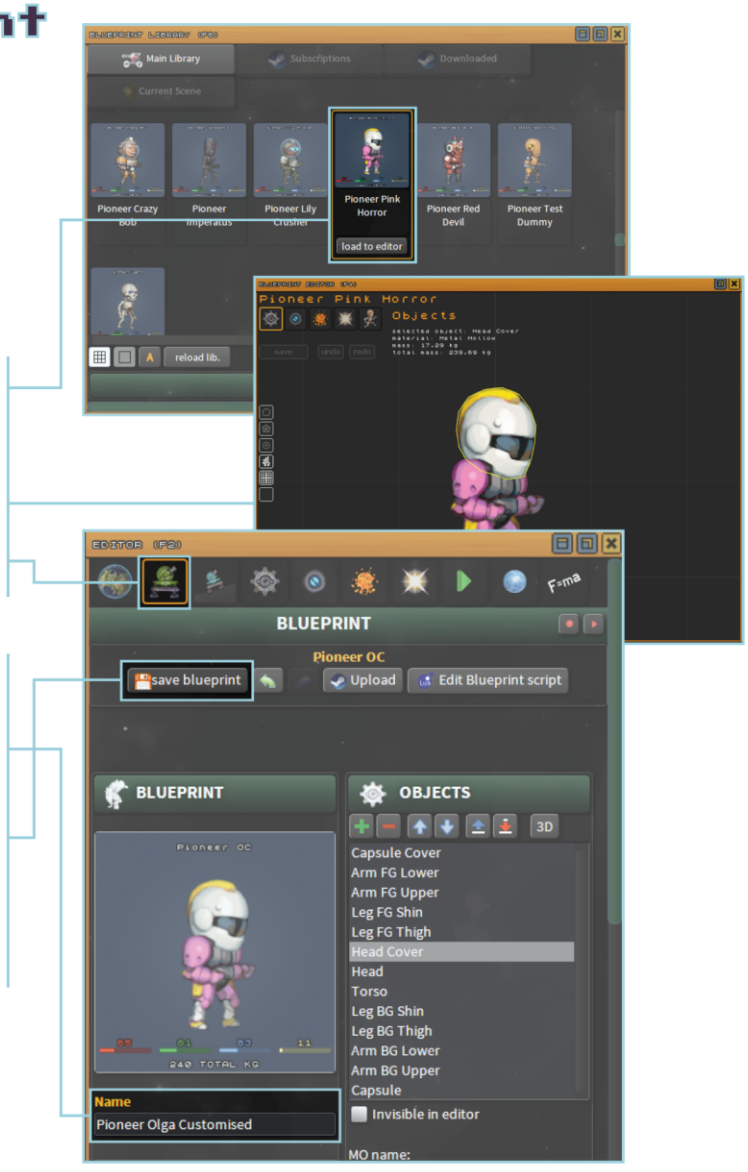
While in *Blueprint* tab you **can't make changes to assemblies** already in the scene. However, you can add **temporary instances of the active blueprint** by clicking somewhere in the scene and pressing the ***Insert*** key.

Customisable blueprint

Modifying a blueprint affects all of its instances in all the scenes it's been added to, so be careful with them. **Do not save changes to the original assets**. Instead, create your own copies. As an example, create your own pioneer.

Open ***Blueprint library*** (*F6*) and **double-click *Pioneer Pink Horror*** (not Bob since he has some personalised voice clips). This opens a new window, ***Blueprint editor***, with the selected blueprint in it and moves to ***Blueprint* tab**.

Create a copy by giving the blueprint a new **unique name** (starting with the word "***Pioneer***") in *Editor* window's *Blueprint* tab, under ***Blueprint* panel.** Once the name's changed, press ***Save blueprint*** on top of ***Editor* window** (or *Save* on top left of *Blueprint editor* window). You now have your own copy of the blueprint.



Blueprint tab: modifying textures

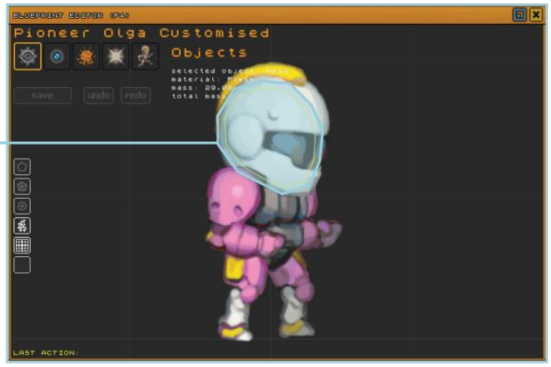
In the *Blueprint* tab **the active blueprint's textures can be changed**. To test this, set a custom head for your pioneer.

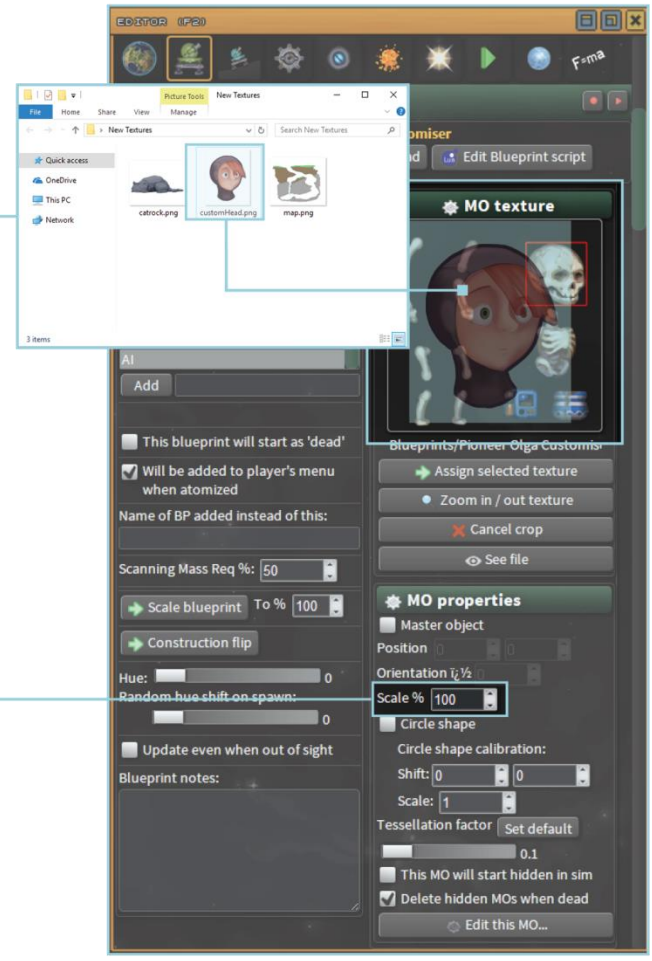
Find a texture you'd like to use, anything goes as long as it's a **PNG with a transparent background**. Once you have it ready, select the pioneer's ***Head* MO** by clicking it in ***Blueprint editor*** (it's covered by the helmet, so you may have to click twice).

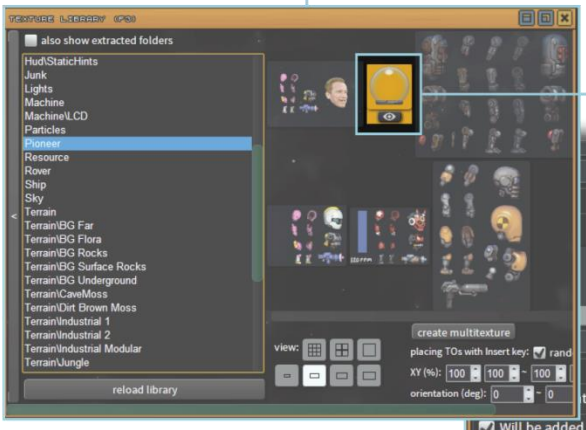
With the head selected, **drag the new texture** in on top of ***MO texture*** in *Editor* window's *Blueprint* tab's ***Objects* panel**. The texture gets replaced but might not be a perfect fit. Change its **size** by adjusting the ***Scale* %** value under ***MO properties***.

There's one problem: the helmet is blocking the results of your work; replace it with a transparent one. Select the ***Head Cover* MO** and open ***Texture library*** (*F3*). Go to the ***Pioneer*** folder, locate the **helmet texture** and drag it to *Editor* window's ***MO texture***. **Scale** the helmet to fit. **Don't forget to save**.

You can make similar changes to any blueprint. Just make a copy of the original one first.







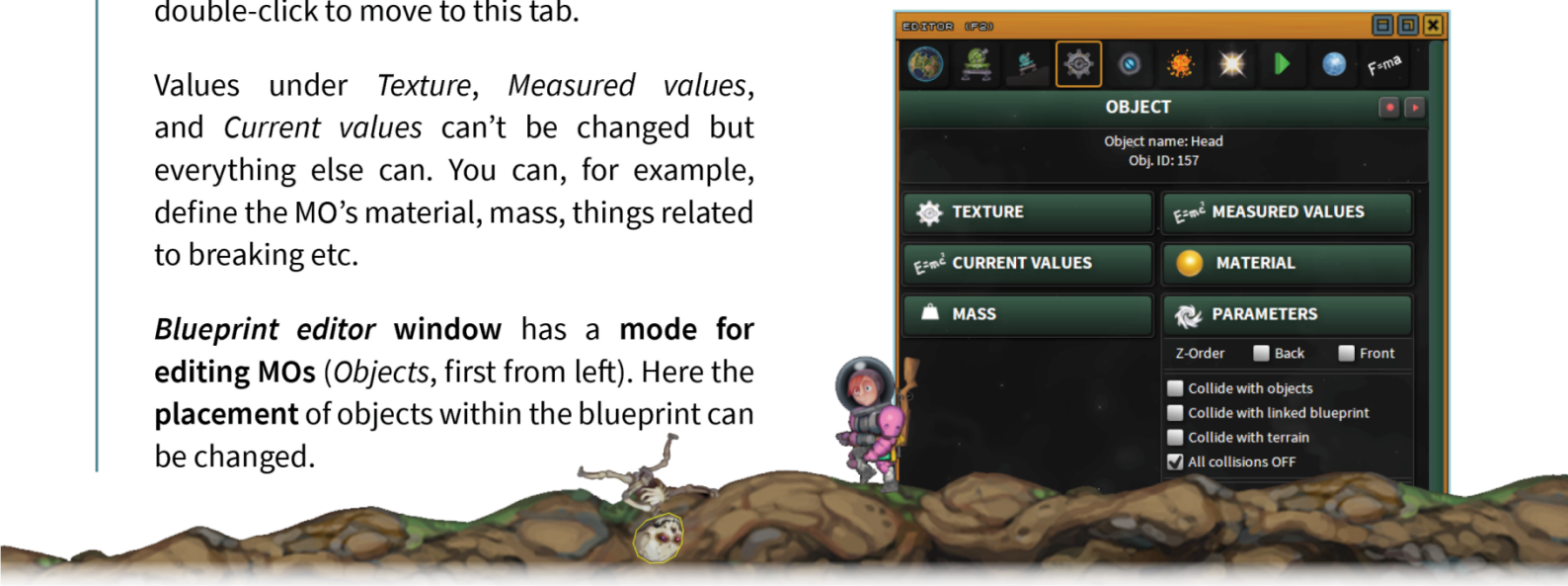
Objects

***Object* tab** (4th from left, *4*, 4th icon after *Edit* in the toolbar) lets you **adjust values assigned to individual MOs**. Active blueprint's MOs are listed under *Blueprint* tab's ***Objects* panel**. Click an MO to open its values in *Object* tab, double-click to move to this tab.

Values under *Texture*, *Measured values*, and *Current values* can't be changed but everything else can. You can, for example, define the MO's material, mass, things related to breaking etc.

***Blueprint editor* window** has a **mode for editing MOs** (*Objects*, first from left). Here the **placement** of objects within the blueprint can be changed.

In *Object* tab **changes can be made to existing assemblies**. Select an MO in the **scene** (**not** in *Blueprint* tab or *Blueprint editor*) and test what changing values does. Remember that these changes will not be permanent.



Objects: adding a new MO

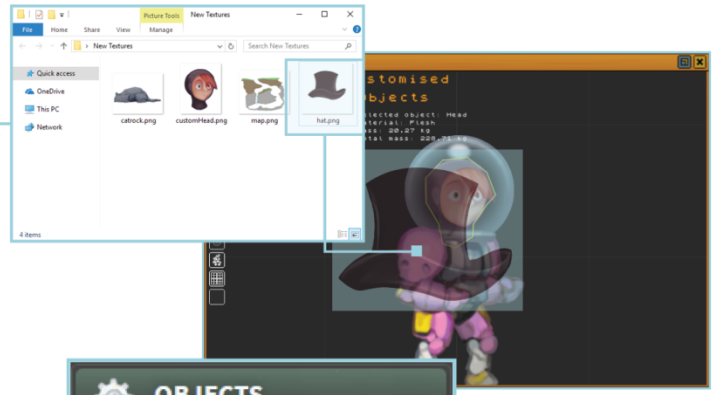
To familiarise ourselves with modifying MOs, let's start with a classic: **adding a hat**.

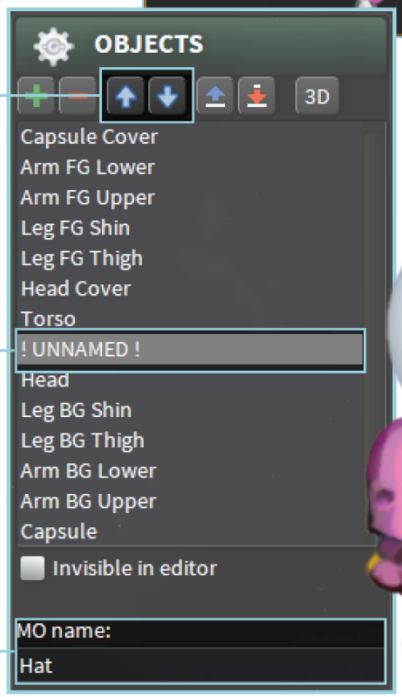
Find an image you'd like to use. If it isn't already, **load your pioneer to *Blueprint editor***. Locate the image on your computer and **drag it into *Blueprint editor* window**.

The text "***MO name undefined***" will appear. Go to ***Editor* window**, **under** *Blueprint* tab's ***Objects* panel** type a logical **name** for the new unnamed MO ("***Hat***").

Like before, change the hat's **size** with ***Scale*** % in ***MO properties***. **Move** it to the correct position in ***Blueprint editor*** by **dragging**.

If the hat is behind the helmet and you want to **bring it to front**, select the hat MO in *Objects* panel and press the **light blue up arrow** above the list until it's on top. For the **opposite**, press the **light blue down arrow**.





Objects: changing MO values

Continuing with the same example, move to ***Object* tab** with the hat selected.

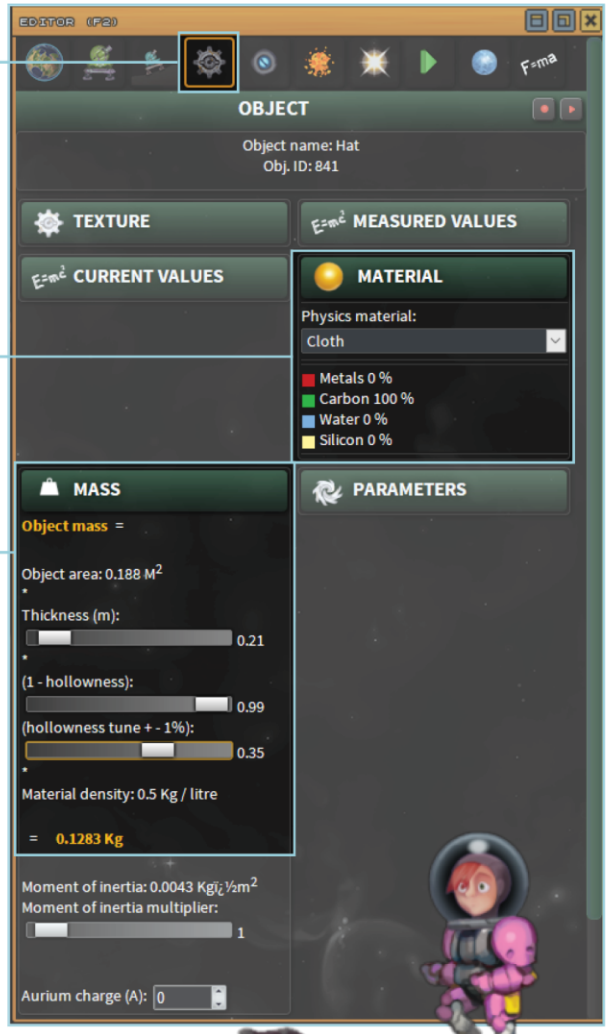
In ***Material* panel** select a physics material for the hat, ***Cloth*** is probably closest to what you want. (For atomisable MOs material affects what resources you get from them.)

Move to ***Mass* panel**, the top three sliders are what we'll be looking at. Even though there are three sliders, the **most important thing is the resulting value of mass** (the yellow number below them). A hat could be around 80 to 140 grams so **0.08 to 0.14 kilograms**, try to aim for that.

You can change other values as well if you want, but for this guide material and mass will be enough.

Once you're done with *Object* tab, move back to ***Blueprint* tab**. **Click somewhere in the scene and press the *Insert* key** on your keyboard. This adds a (temporary) copy of the active blueprint to your scene.

You probably see a problem: **the hat doesn't stay on**. This can be fixed with **joints**.



Joints

Joints keep blueprints' MOs together and **MOs can rotate around these joints**. Values related to them are adjusted in ***Joint* tab** (5th from left, *5*, 5th icon after *Edit* in toolbar).

Joints can also be used to **attach assemblies to TOs or other assemblies**; the plus signs mentioned in the interactivity section are actually joints.

Blueprint editor* window** has a **mode for editing joints** (2nd from left). New joints can be **added** and existing ones **moved** and **rotated**. The ***3D view button shows the MOs and their connecting joints in 3D space.

Like with *Object* tab, **changes can be made to assemblies** in *Joint* tab. Go to *Joint* tab, select a joint in the scene and see what different values do.

Joints: attaching an MO

With your pioneer still in ***Blueprint editor***, move to its ***Joints* mode**.

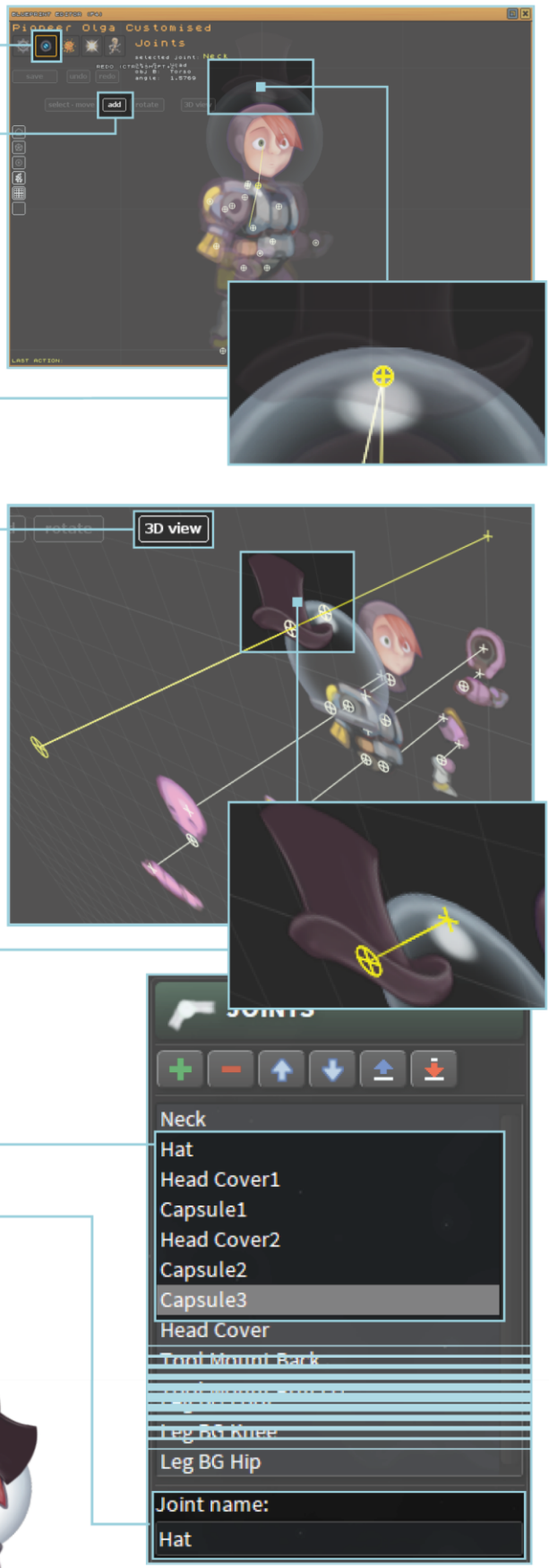
Click the ***Add*** button, now you can add joints by clicking where you want them. Make sure that there's at least some **overlap between the hat and the head cover** and **add a joint between them** by clicking.

Click ***3D view***. You'll see that you got more joints that you wanted. **The engine creates joints between all the overlapping MOs** (even though you can't see them, there are two capsule MOs connected to the pioneer).

Next you'll have to **remove all the unwanted joints**. Go to ***Blueprint* tab's *Joints* panel**. The ones you just created have automatically generated names, probably along the lines of "*Hat*", "*Head Cover1*" or "*Capsule1*". Note that the automatically generated *Hat* isn't necessarily the correct one and may have to be removed.

Click one of the listed joints and look at *Blueprint editor*'s 3D view. If the joint **highlighted with yellow <u>is</u>** where you **placed** it but **<u>isn't</u>** the one **between the hat and head cover, delete it** by pressing the *Delete* key. Do this until the only one that remains is the **one between hat and head cover**. Once you're done, give it a **new, logical name below the joint list** ("***Hat***").

Click somewhere in the **scene** and press ***Insert***. The hat stays on but **rotates around its joint**; the joint's values need to be edited in ***Joint* tab**.



3D view controls

Move the view with *RMB*, rotate it with *LMB* and zoom with mouse wheel. Exit 3D view by clicking one of the other buttons (*Select + move*, *Add*, or *Rotate*.)

Joints: changing joint values

Select the new ***Hat* joint** and move to ***Joint* tab**. Here we'll be looking at two panels: ***Friction*** and ***Angular spring***. There are no absolute correct values for these, it's better to just try and see what looks best.

Friction affects **how easily an MO rotates around its joint**. **Set the value above zero** to keep it from rotating uncontrollably.

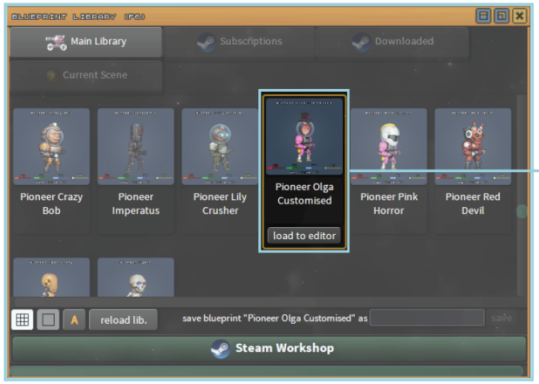
Angular spring holds values for *Angular spring acceleration* and *Angular spring angle*. **Angle** sets the **desired angle for the joint, acceleration** affects **how strongly the joint tries to reach that angle**. **Angle can be left at zero** if you haven't rotated the hat and are happy with how it looks.

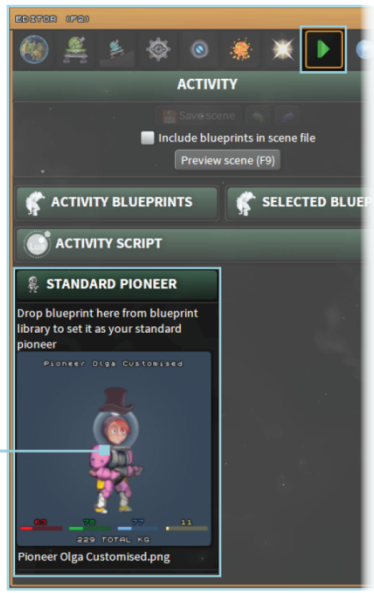
Acceleration should be adjusted as without it the hat won't make any attempt to keep its angle. The numbers given here can sometimes get quite large, most of the pioneer's other joints, for example, have a value of 6000.

Once you're done your pioneer has a new, stable hat. **Remember to save!**

Replacing standard pioneer

Your pioneer is now ready to go. To make it the **standard pioneer** to be used in all the scenes go to ***Activity* tab** and drag your pioneer from ***Blueprint library*** (*F6*) to ***Standard pioneer* panel**. Save the scene.



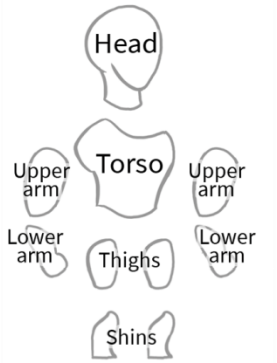


Joints that stick to TOs or assemblies

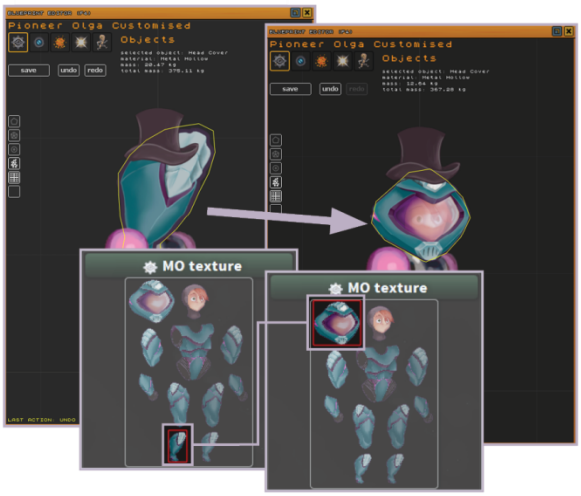
To make a blueprint's joints attachable to TOs or other assemblies, open the blueprint in editor and go to *Blueprint* tab's *Joints* panel. There click the joint you want to change and below the list check the fitting checkbox(es) (*Auto Pin to Objects*, *Auto Pin to Terrain* etc.)

Pioneer with entirely new texture

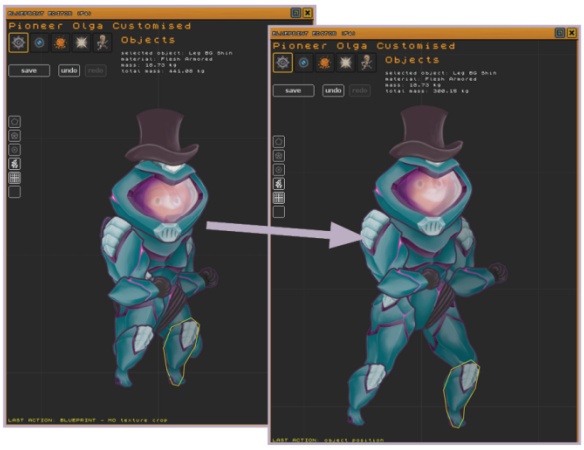
Use a texture that consists of the **same parts as the existing pioneers**. The proportions or shapes don't have to be the same, MOs and joints just need more tweaking the more they differ. All the pieces can be **part of the same** (transparent) **PNG as long as they don't touch**; the engine knows how to crop it appropriately.



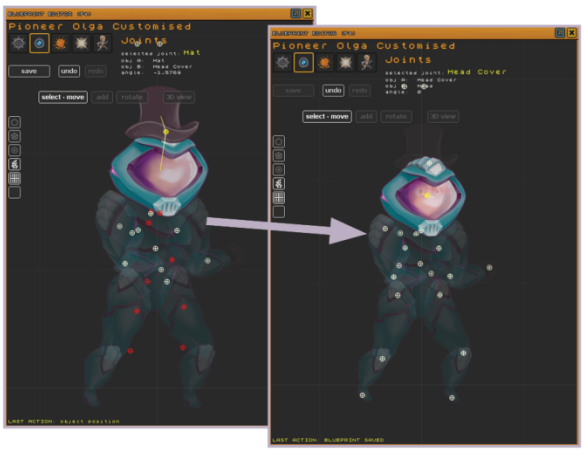
Select an MO in *Blueprint editor* and drag the new texture into *Editor* window's *MO texture*. Since it consists of several pieces, you'll have to **select the correct one**. Do this **by clicking it in *MO texture*** (the one outlined with red is the one in use). Scale the texture with *Scale %* under *MO properties*. **Repeat for every MO**.



Once you're finished, depending on how much your texture's proportions differ from the original ones, the pioneer might look a bit funny; you may have to **reposition MOs** in *Blueprint editor*'s ***Objects* mode**.



Once the MOs are in place, go to ***Joints* mode**. If you didn't add any completely new MOs, no new joints need to be added. However, the existing ones probably aren't where they're supposed to; the **red ones are in the wrong places** for sure, but it's best to **check the positions of the other ones as well**. Names listed under *Blueprint* tab's *Joints* panel help you determine where each should be (*Knee* on top of knee etc., *FG* stands for foreground, *BG* for background).



Finish that and your pioneer has a new custom texture. **Save**.

Blueprint from scratch

This guide won't go into detail on how to create completely new blueprints. However, if you're interested in doing that, at the bottom of *Blueprint* tab click *! clean up blueprint, create new*, give it a name and start adding components (MOs, joints etc.) Note that blueprints need to have a master object. Set it by checking *Master Object* in *MO properties*.

Troubleshooting

Accidentally created a TO/MO when trying to replace a blueprint's texture
» Delete the unwanted object and try again; make sure you drag the texture to *Blueprint* tab's *MO texture*.

Accidentally replaced wrong part of the pioneer's texture
» Either press *Ctrl* + *Z*/*Undo* in *Blueprint editor* to undo the changes or locate the original texture in *Texture library* (*F3*) and replace the texture again with the original one.

Accidentally saved on top of the original blueprint
» This doesn't *necessarily* break anything, in general it's just better to keep the original assets as they are. To recover from this:
  » **If you want to keep both versions**, first save the changed version with a different name. Next go to *Blueprint library* and find the one that is supposed to be the original (the one with unchanged name). Load it to *Blueprint editor* by double-clicking. In *Editor* window's *Blueprint* tab scroll to bottom; there you'll see *Backup versions* panel. Double-click the oldest (top) entry on the list to load the original version. Click *Save*/*Save blueprint*. The blueprint has been restored to its original state.
  » **If you *don't* want to save the changes** you've made, in *Editor* window's *Blueprint* tab scroll to bottom to *Backup versions* panel. Double-click the oldest (top) entry on the list to load the original version. Click *Save*/*Save blueprint*. The blueprint has been restored to its original state.

Custom head doesn't stay in place
» If the texture differs a lot from the original one, it might not reach the joint that should keep it in place; there should be at least some overlap between the head and the torso. Move the head so that it is on top of the torso and/or check the placement of the neck joint in *Blueprint editor*'s *Joints* mode.

Nothing happens when *F9* is pressed/can't move the character
» Press *P*, the simulation might be paused
» Check that you're in game mode, simulation can be on even in edit mode

Pressing the *Insert* key adds hats/other individual MOs to the scene instead of an assembly
» This means you're in a wrong *Editor* window tab. Move to *Blueprint* tab to add test assemblies.

Pressing the *Insert* key adds an MO to *Blueprint editor*, not an assembly to the scene
» Undo the changes in *Blueprint editor* and click somewhere in the scene; make sure no windows are active (active windows are outlined with orange) and try again.

Pressing *F9* in game mode doesn't reset the scene
» If you've added multiple pioneers to the scene (even if they're temporary), instead of resetting, *F9* makes you leave the current pioneer and (try to) move to another. Go back to edit mode (*F2*) and reset there.

The blueprint loaded to *Blueprint editor* doesn't show up in *Blueprint* tab
» Try moving to game mode (*F2*) and back.
» Click somewhere in the scene and press *Insert*.
» Try reloading it to editor through *Blueprint library* (*F6*).

Now you should know how to build scenes and add and edit blueprints. Lastly, we'll discuss how to use different effects. In this part particle systems and lights will be introduced.

Particle systems

Particle systems (PS) can be used in place of **simple, repeating animations**. They consist of **small, physics-based** objects, **particles**, which are created, or **emitted**, according to the values given in *Particle system* tab.

In *Planetoid Pioneers* particle systems are used for different things: they bring life to the world with **small visual effects** that can be attached to either **blueprints** or the **terrain** itself. For example, the stars you see when a character is stunned are particles.

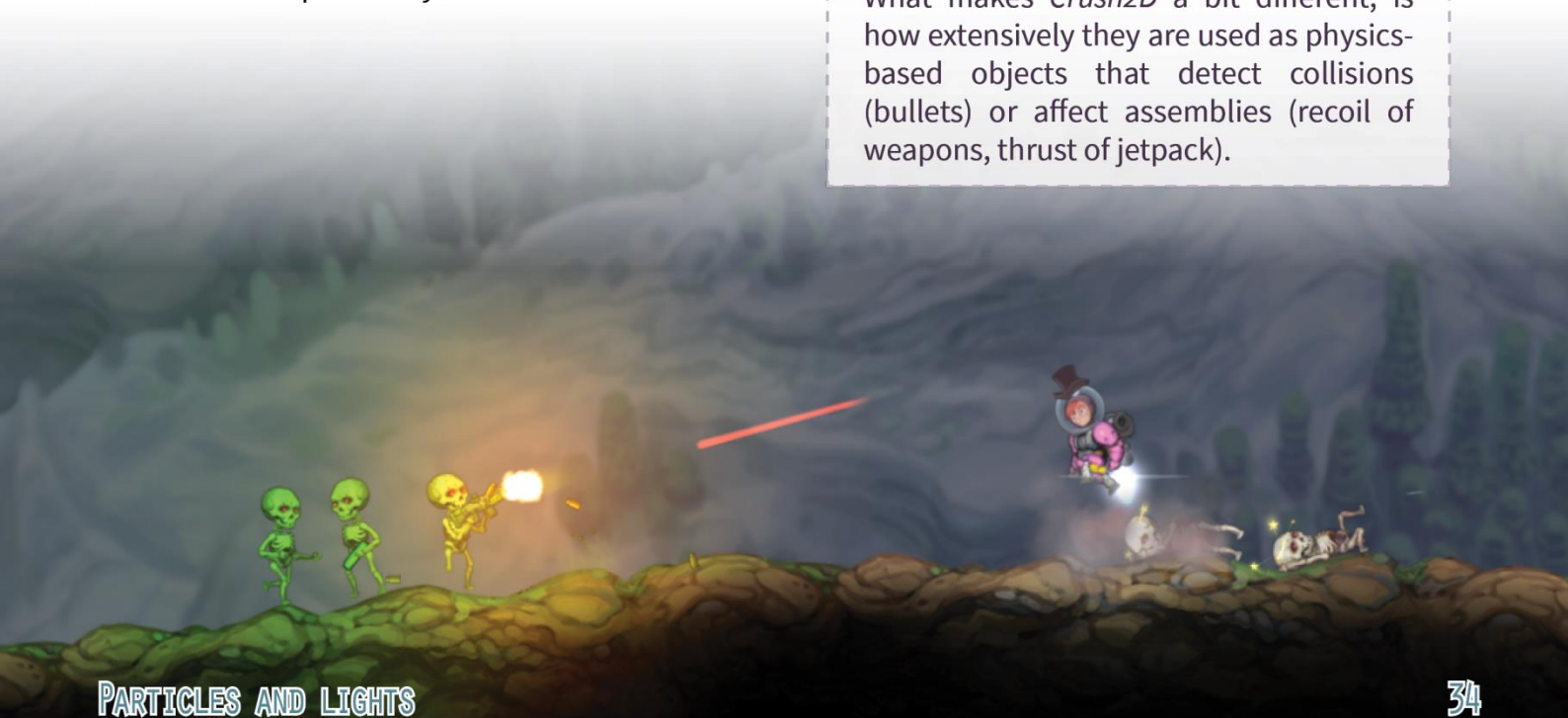
Besides just looking nice, they can also have **physics-based functionality**. In *Planetoid Pioneers* all the weapons' bullets and the jetpack's flames' thrust force, for example, are made with particle systems.

To make more intriguing and complex effects, it's possible to **add particle systems to particles** themselves. These are called **child PSs** (making the "upper" PS their "**parent**").

Since particle systems' opportunities are basically unlimited, there definitely is no one correct way of implementing them. This guide only introduces the tools you have available for customising them.

About particles

Particle systems aren't a new invention, as several games engines offer a similar possibility to make particle-based effects. What makes *Crush2D* a bit different, is how extensively they are used as physics-based objects that detect collisions (bullets) or affect assemblies (recoil of weapons, thrust of jetpack).



Particle system tab

***Particle system* tab** (6th from left, *6*, 6th icon after *Edit*) is where **particles are modified**.

***Child PSs tree* panel** gives an **overview of the PS and its child PSs**.

***Particle* panel** defines **what kind of particles** will be emitted. ***Trail* panel** sets values for **trails left by them**. In ***Glow* panel glowing effect can be added** to the particles.

Emitting* panel** specifies how the particles act **during creation**. ***Bounce* panel** affects how they act when they **hit something**. ***Particle update defines how particles act **after they have been emitted**. There are other panels as well, but these seven are what we'll be focusing on in this guide.

When in this tab, you can see the PSs present in the scene. The **squares linked by lines** define the **starting points** for particles; the **arrows** show the **directions** they move in.

Important! At the bottom there's a button *Save this PS to PS library*. **Do not** press that unless you know what you're doing. Check this guide's *Saving to PS library* for instructions on how saving to the library works.

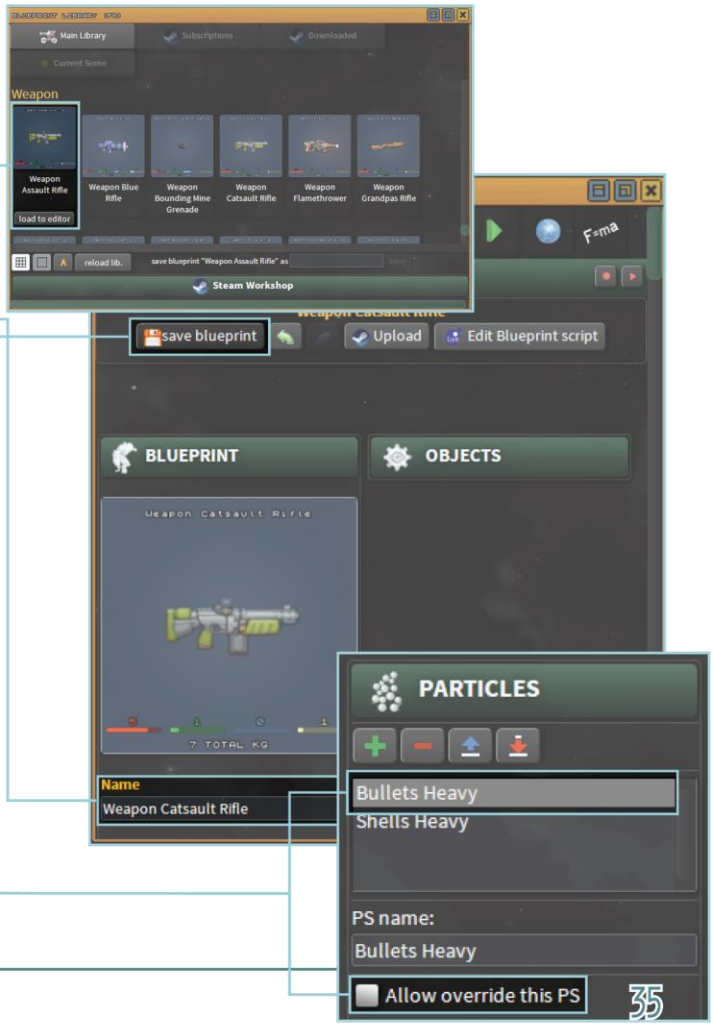
In *Particle system* tab **temporary changes** to particles present in the scene **can be made**. What makes it unique is the fact that **there is a way to save these changes by copying and pasting**, which is a feature utilised in the next example.

Customisable particles

Let's start by customising a blueprint's particles. Open ***Blueprint library* (F6)**, find ***Weapon Assault Rifle*** and load it to editor.

Like before, you don't want to make changes to existing assets. **Make a copy of the weapon by renaming** (start the new name with the word "***Weapon***") it in *Blueprint* tab's *Blueprint* panel and **saving**.

In the same tab, take a look at the ***Particles* panel**. You'll see that there are two PSs attached to the blueprint: one for bullets, other for shells. Select ***Bullets Heavy*** and **uncheck *Allow override this PS***.



Particles panel

No need to do any of these now, but here you can also add and remove PSs (+ and - icons) and change their position (*Link pos.*) and orientation in relation to the blueprint.

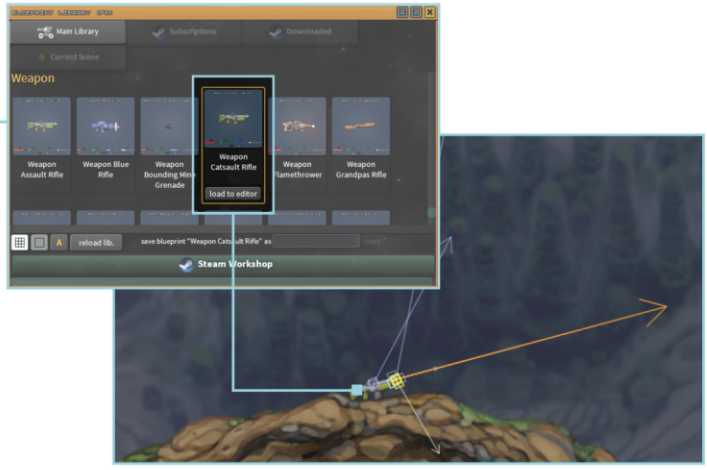
Exploring existing PSs

Move to ***Particle system* tab**. Open ***Blueprint library*** (*F6*) and **drag your weapon to the scene** to create a temporary assembly. Select the assembly's ***Bullets Heavy*** particle system.

Under *Particle system* tab's ***Child PSs tree* panel** you can see that the PS has six child PSs, some of them children of children.

You can **test** what they do by **selecting one** and checking ***Emitting*** on top of *Editor* window (note that emitting a parent PS can also cause a reaction in its children).

If all these child PSs seem too confusing, you can **remove** some of them with the **red minus sign** above the list. **The most important one is *Tracer Bullet***, so leave at least that. Check the **table below** for short descriptions of each PS to see which ones you want to keep.



Selecting the correct PS

If there are a lot of PSs on top of each other in the scene, it can be difficult to select a specific one. This is where *Assembly* tab comes in handy. Move to *Assembly* tab, select the assembly the PS is attached to, and under the tab's *Particles* panel double click the PS you want to edit.

About child PSs

The PSs marked with three dots are direct children of the main PS, the ones with more dots are children of children. How child particles act in relation to their parent is defined in *Child PS emit* panel. They can, for example, be emitted at the same time with their parent or whenever a parent "dies".

PS	Parent	Description
Bullets Heavy	-	Main particle system, can't be removed; has recoil and a smoke effect
Light	Bullets Heavy	Large light that flashes with every shot
Muzzle Flash	Bullets Heavy	Smaller light that flashes with every shot
Tracer Bullet	Bullets Heavy	Bullets, don't remove
Bounce Tracers	Tracer Bullet	Particles that are created where a bullet hits
Smoke Trail	Tracer Bullet	Smoke that rises from the pipe of the weapon
Smoke	Bullets Heavy	Smoke that shoots forward from the pipe of the weapon

Editing particle appearance

With the **assembly's** *Bullets Heavy* still in *Particle system* tab, select ***Tracer Bullet*** in *Child PSs tree* panel. Check ***Emitting*** so you'll see the changes in real time.

Under ***Particle* panel's *Particle texture*** you can set any texture from the **library** (F3) or import a new one from your **computer**. This is how the particles themselves will look. In the same panel there's a section, ***Particle color***, where you can set two colours between which **the colour of particles will be generated**. You'll also probably want to **change the alpha value**, as 0 makes the particles invisible.

Moving on to ***Trail* panel**. First of all, if you want your particles to have trails, make sure that ***Emit particle trail*** and ***Render trail in scene*** on top are **checked**. Like with particles, you can set **texture** and **colour** for your trail. At the bottom there are options for trail **length** and **width**.

Finally to **glow** effects. First, at the top of *Glow* panel **check *Render glow in scene***. Set **texture** and **colour** for the effect. With ***Glow size*** you can set two values between which **size of the effect will be generated**.

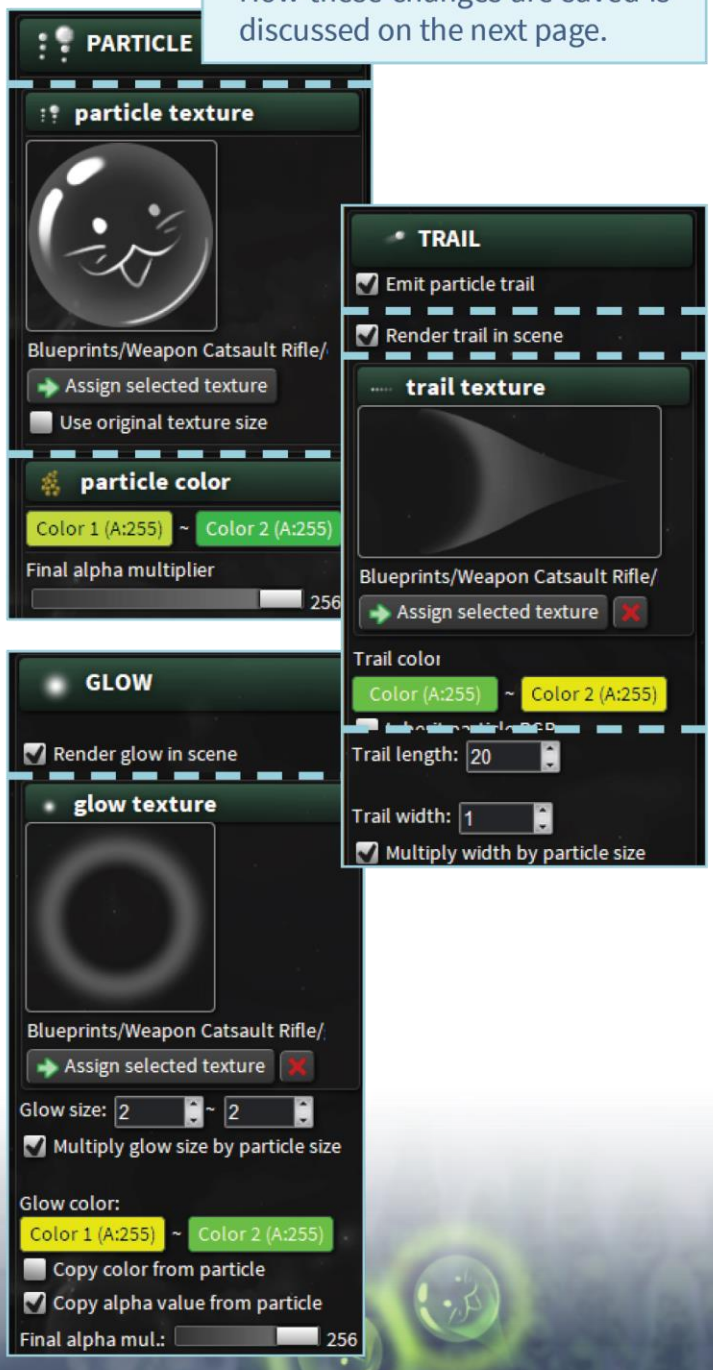
There are other options in all three of these panels as well. You are, of course, free to try them out too.

Hint

It might be difficult to see the particles since they're moving so fast. We'll come back to emitting options shortly, but for now you can set *Emitting* panel's *Start speed* a lot smaller.

Important!

Remember that you're still making changes to an assembly and they **will be lost if you reset the scene, so do not do that**. How these changes are saved is discussed on the next page.



Checkpoint: saving the changes made so far

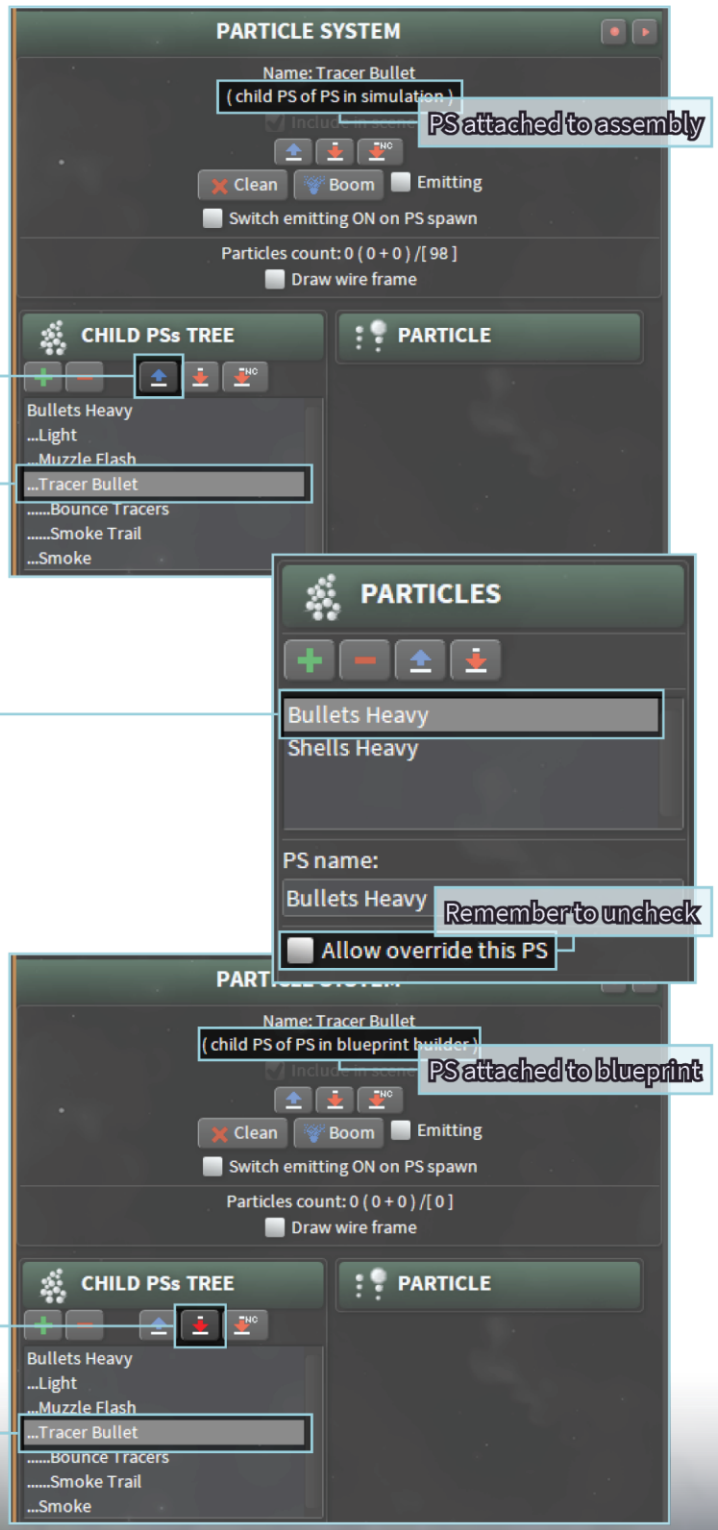
This is where the previously mentioned **copying and pasting** is needed.

Once you're happy with how your particles look, with the assembly's *Bullets Heavy's* ***Tracer Bullet*** still **selected**, click the **blue up arrow** in *Child PSs tree* to **copy** it.

Next, if it isn't there already, **load your custom weapon to editor**. Make sure this is the blueprint active in *Blueprint* tab.

In ***Blueprint* tab's *Particles* panel double click *Bullets Heavy*** to bring it to ***Particle system* tab**. There, select ***Tracer Bullet*** in *Child PSs tree* and press the **left red down arrow**. This **pastes** the changes to the currently selected (child) PS.

Finally, move back to ***Blueprint* tab** and click ***Save blueprint***. The changed PS is now saved as a part of your blueprint.



Pasting without replacing

If you don't want to replace one of the existing particle systems, you can create a new child PS with *Child PSs tree* panel's green plus sign and then paste the copied particles there.

Editing particle behaviour

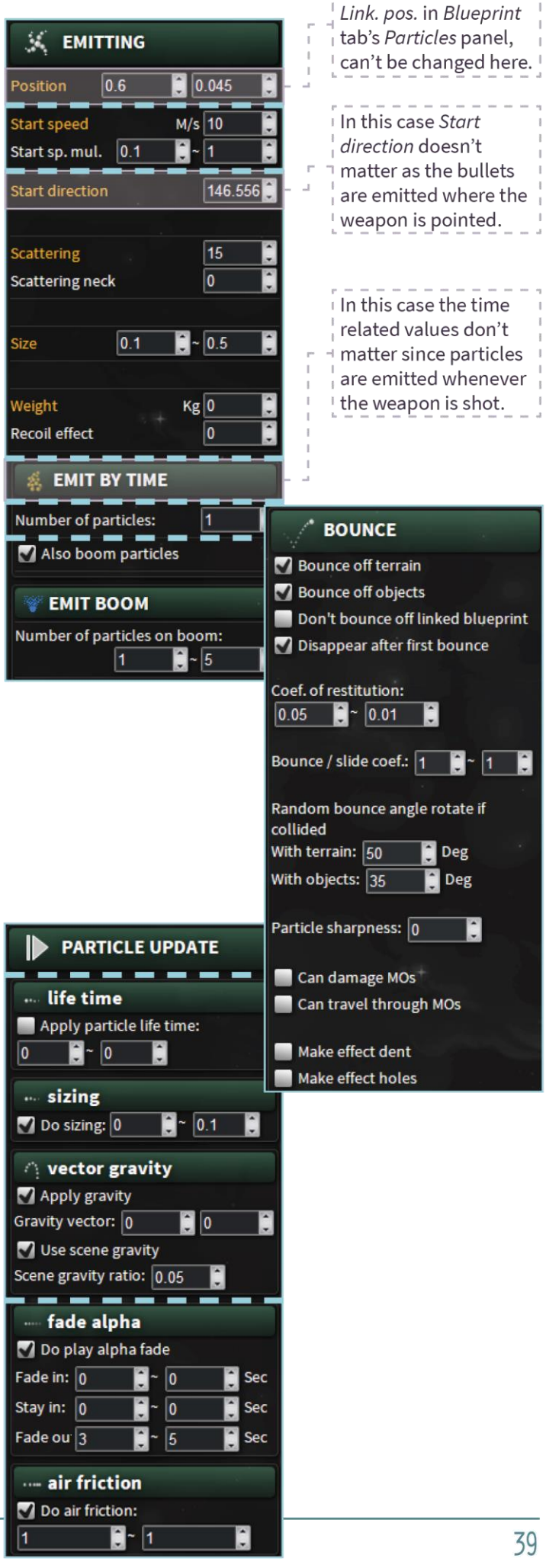
Bring the **assembly's** *Bullets Heavy* back to *Particle system* tab (if you already reset the scene, create a new assembly). Select ***Tracer Bullet*** again and check ***Emitting*** on top of the tab.

In ***Emitting* panel** set ***Start speed*** for the particles; use ***Start sp. mul.*** to add **variation**. ***Scattering*** affects the **spread** of particles (angle from 0 to 360). ***Scattering neck*** affects how often particles are emitted **towards *Start direction*** despite *Scattering*. ***Size*** defines two values between which a **size for particles is generated**. You can also set ***Weight*** for them. ***Recoil*** makes the connected MO **"jump"** when particles are emitted. For **more than one particle** in each shot, either change the value in ***Number of particles*** (fixed amount) or ***Number of particles on boom*** (random amount, check ***Also boom particles***).

In ***Bounce* panel** use the **first four checkboxes** to set **conditions for bouncing**. ***Coef. of restitution*** (0 to 1) affects **how much the particles bounce**. ***Bounce / slide coef.*** (0 to 2) and ***Random bounce angle rotate if collided*** (0 to 180) affect the **direction the particles bounce in**. ***Particle sharpness*** **and the four checkboxes after** that affect the **damage** done by them.

In ***Particle update* panel** you can use ***Life time*** to make the particles **disappear after certain time**. ***Sizing*** is used to **grow** (positive values) or **shrink** (negative values) the particles over time. ***Vector gravity*** can be used to make the gravity affect the particles. With ***Fade alpha*** particles can be **faded out** after a certain time. ***Air friction*** affects how **effortlessly particles move through the air**.

Again, there are a lot of other options as well. Nothing's stopping you from trying them out.



Finishing the custom weapon

Once you're done, go through the same steps as you did after editing the particle appearance to **save the changes**: copy the assembly's *Tracer Bullet*, paste it on top of the blueprint's matching PS and save the blueprint.

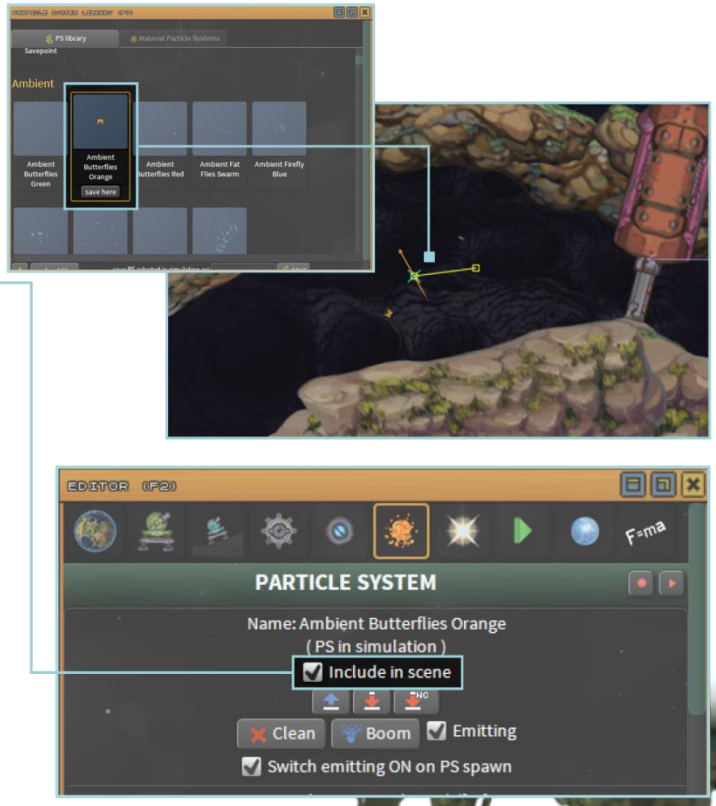
If you want, you can customise the weapon's other (child) PSs as well using the same logic. You can also give your weapon a custom texture following the instructions in this guide's *Customising blueprints* section.

Adding particles to scenes

Particle systems can also be saved as parts of scenes. Press ***F7*** (or the 8th icon after *Windows* in the toolbar) to open ***Particle system library***.

You can **drag** any of these into the scene. To make a PS part of a scene, check ***Include in scene*** at the top of ***Particle system*** **tab**. Next, move to a **tab where the scene can be saved** (*Scene* tab, for example) and click ***Save scene***.

Like before, you can **modify** this PS in *Particle system* tab. Since it's saved as a part of a scene, it doesn't have to be copied and pasted anywhere; just **save the scene again once you're done editing**.



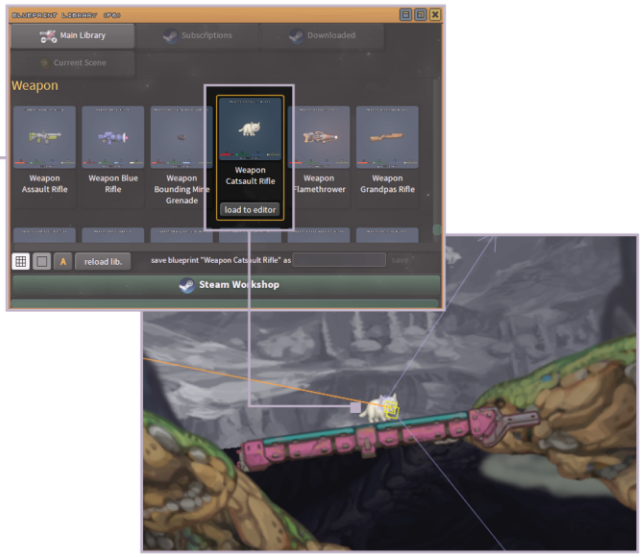
Particle layers

Like TOs (see page 13), PSs use layers. Their collisions are detected only on layers -3 to 3. The layer of a PS is defined by *Render in LAYER* in *Particle* panel.

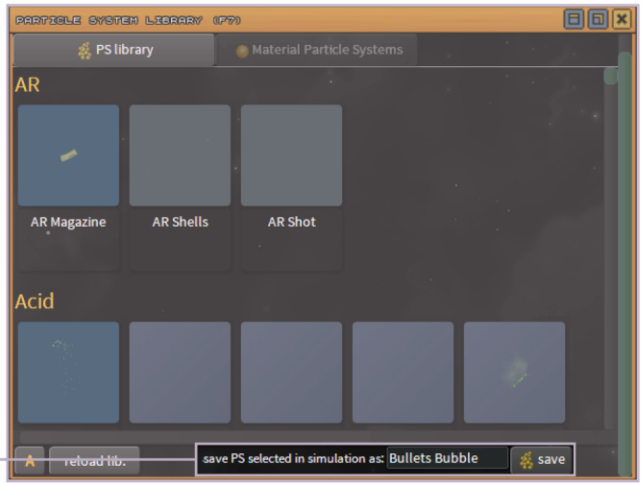
Saving to PS library

You shouldn't carelessly click the *Save this PS to PS library* button because it **overrides the (main) PS with the same name**. A bit like blueprints, these ready-made PSs can be used in several places, so **saving on top of an existing PS can have a bigger effect than you might expect**.

Saving all the PSs to the PS library is not necessary, as customised PSs can be saved as parts of blueprints or scenes (your customised *Bullets Heavy*, for example, is a part of your weapon even if it isn't in the PS library). This feature becomes **useful if you want to use the PS in other places** as well.



To save the custom *Bullets Heavy* PS, first **add your weapon to the scene** (temporary or permanent copy, doesn't matter). Move to ***Particle system* tab** and select the assembly's ***Bullets Heavy***.



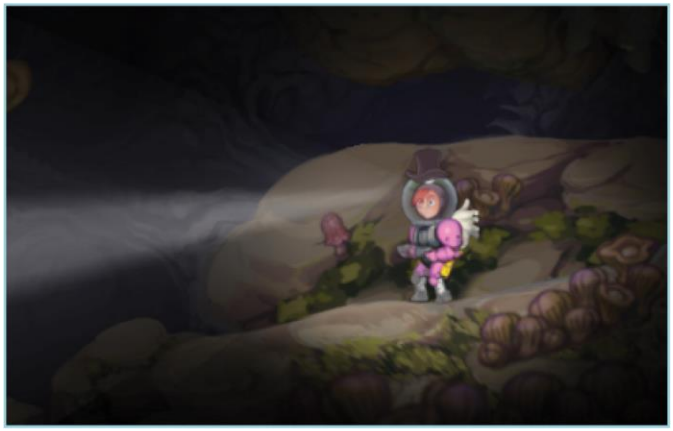
Open ***Particle system library*** (*F7*). With your PS still selected, locate the textbox labelled ***Save PS selected in simulation as*** at the bottom of the window. Type a **unique name** for your PS and click ***Save***. Your PS is now part of the library. If you want to make changes to this PS, you can save them with the PS tab's *Save this PS to PS library* button.

Note! The **PS attached to your weapon** is still a customised version of *Bullets Heavy*, **not an instance of the one you saved to the library**. Editing the one attached to the blueprint won't change the one in library or vice versa.

This step is not necessary, but if you want the blueprint to get its PS **from the library**, **load it to editor** and in *Blueprint* tab's *Particles* panel select ***Bullets Heavy***. Check ***Allow override this PS*** and after ***Override by*** type the **name of your PS** (note that this literally overrides the PS, so from now on changes made to the blueprint's *Bullets Heavy* will be ignored).

Lights & Light tab

Before finishing this guide off, let's discuss a bit simpler effect type: lights. They are, as the name implies, **sources of light** that **can be added to blueprints and scenes**.



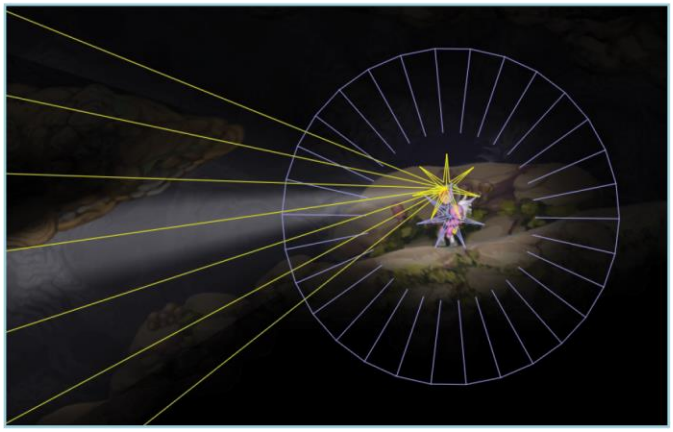
At this point it's good to check that you have **lighting toggled on** (toggle with *L*). Also **switch on discovery map** (*K*).

Planetoid Pioneers has **two kinds of lights**: circle and spot. **Circle** lights emit light in a **circular shape**; **spot** lights form a **"cone"** of light pointed at the specified direction.

For an example of lights, take a look at the pioneer: it has one as a headlamp (spot light) and other, a bit fuzzier, around the pioneer itself (circle light).

In ***Light* tab** (7th from left, *7*, 7th icon after *Edit* in the toolbar) **new lights can be added to the scene** and the **existing ones edited**. While in this tab, you can see all the lights present in the scene.

In this tab **temporary changes can be made**. However, there are **differences between assembly and scene lights**.



Like with MOs and joints, **you can select any light attached to an assembly and test what changes do**. They will be reset with the scene and **there is no way to save them**. Permanent changes have to be done through a blueprint.

Changes made to **scene lights** work a bit differently. Unsaved changes are **not reset with the scene**, but will **disappear** if the scene is completely **reloaded**. Permanent changes are made by clicking the *Save scene* button on top.

Editing blueprint lights

Let's carry on with the same example. **Load your custom weapon to the editor**. Its lights are listed under *Blueprint* tab's ***Lights* panel**. There you can see that it has a light, *Muzzle Flash*, attached to it. This doesn't actually do anything, you can ignore or delete it (the red minus sign).

Press the **green plus sign** to **add** a new light. The text *Light name undefined* will appear; under the list **give it a name** ("***Spot***").

After ***Link to MO*** give the name of the blueprint's **MO you want to link this light to**, in this case most likely "***Tool***" (MOs are listed under *Objects* panel). **Move** the light to its desired position with ***Link pos.*** Finally, **double-click the light** to move to *Light* tab.

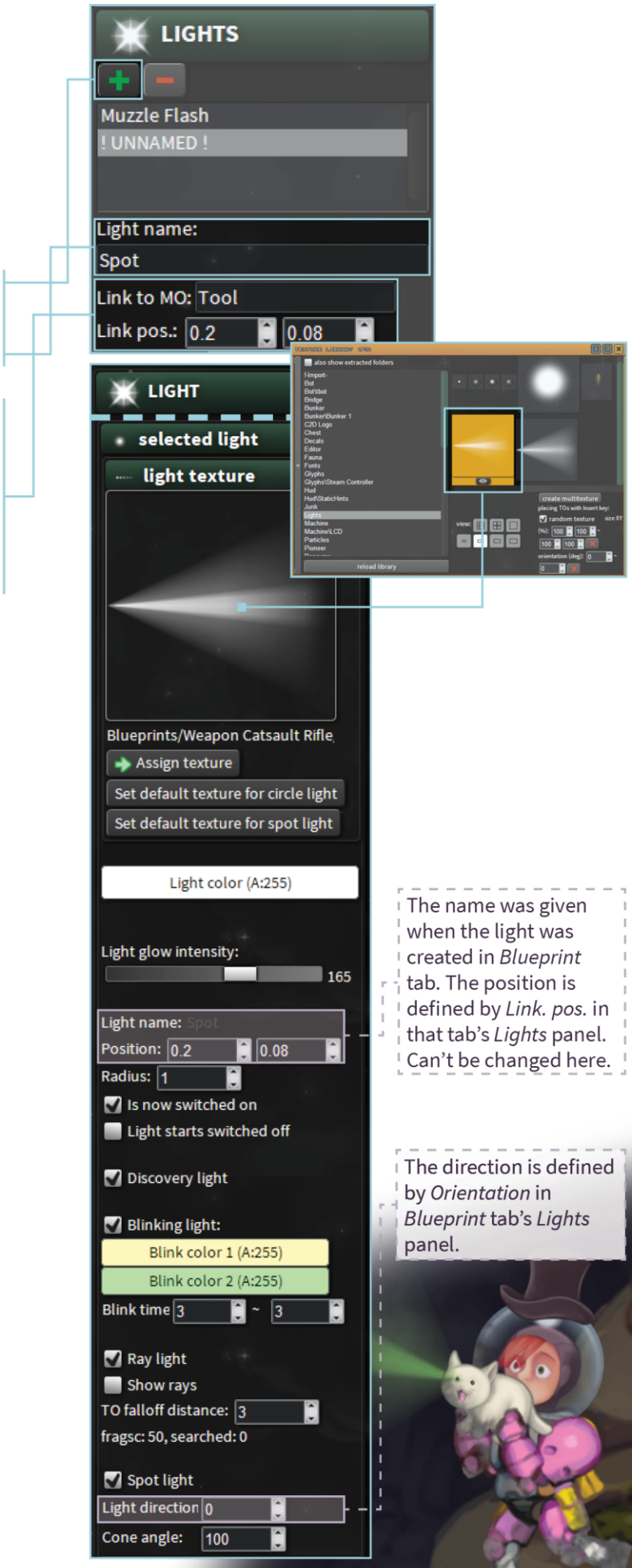
Change the light's **texture**. You can use any image you want, but for this example we'll be making a spot light, so it's **best to use a cone shaped texture**. The one attached to the pioneer's headlamp can be found in ***Texture library*'s** (*F3*) ***Lights* folder**.

There are **two options to setting colour** for the light: either set one colour with ***Light color*** or check ***Blinking light*** and set **two colours between which the light switches**. Use ***Blink time*** to set the **interval** for this.

Light glow intensity sets intensity for the light. ***Radius*** sets the **distance** the light reaches. Checking ***Discovery light*** makes the light **clear the discovery map**.

If ***Ray light*** is checked, the light **can't go through TOs**. ***TO falloff distance*** affects **how far inside TOs** the light can travel.

Lastly, **check *Spot light*** (note that **leaving this unchecked would make the light a circle light**) and **modify *Cone angle*** (value between 1 and 350). **Save** the changes with the ***Save blueprint*** button on top of the tab.



Adding lights to scenes

To add lights to the scene, first go to ***Light* tab**. Next, make sure that the **currently active light isn't part of a blueprint**. Do this by **selecting a light already in the scene** (one attached to the pioneer assembly, for example).

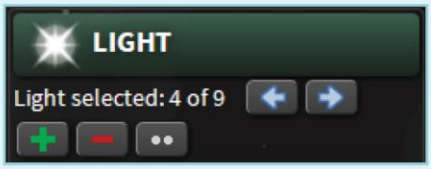
Press the **green plus sign** on top of ***Light* panel** to add a light. You have the same options for editing it as you did with the previous example.

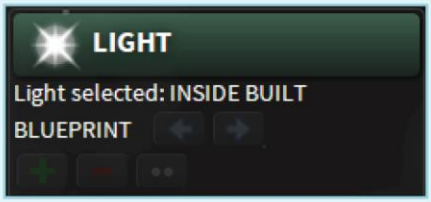
Since this is an independent light, its *Position* and (in case it's a spot light) *Light direction* can be changed. You can also type a name for it after *Light name*.

To **save** the light as part of the scene, click **Save scene** on top of the tab.

In-scene or blueprint light?

The appearance of *Light* tab changes a bit depending on what the currently active light is a part of. If it's an assembly/scene light, the tab displays the text *Light selected: x of y*. You also have options for adding lights to (plus sign) and removing lights from (minus sign) the scene. If it's a blueprint light, you can see the text *INSIDE BUILT BLUEPRINT*. The options are greyed out since blueprint's lights added and removed through *Blueprint* tab





Troubleshooting

Accidentally reset the scene before copying/saving the changes made to an assembly's/the scene's particle system

» Unfortunately there is no way to recover temporary changes. The best way to minimise the risk of losing your work is to every now and then copy the changes from the assembly to a blueprint and save the blueprint (for assembly/blueprint PSs)/save the scene (for scene PSs).

Accidentally saved on top of existing particle system in PS library

» This doesn't *necessarily* break anything, in general it's just better to keep the original assets as they are. The editor does create backups, so it is possible to recover from this. However, these backups can't be accessed through the editor, which makes it a bit more complicated. If you are confident in your computing skills, here's how to recover:

  » Locate the *Planetoid Pioneers* folder on your computer and open *Particle Systems* folder (.../*planetoidpioneers/Particle Systems*)

  » **If you want to keep both versions**, first locate the changed PS in this folder and rename it (right click → *Rename*). Next, move to *backups* sub folder (.../*planetoidpioneers/Particle Systems/backups*). There, find the oldest version of the PS you accidentally replaced, copy it (right click → *Copy*), move back to *Particle Systems* folder and paste (right click → *Paste*). If the name of the backup has an automatically generated ending, rename it using the original name.

  » **If you *don't* want to save the changes**, first locate the changed PS in this folder and delete it (right click → *Delete*). Next, move to *backups* sub folder (.../*planetoidpioneers/Particle Systems/backups*). There, find the oldest version of the PS you accidentally replaced, copy it (right click → *Copy*), move back to *Particle Systems* folder and paste (right click → *Paste*). If the name of the backup has an automatically generated ending, rename it using the original name (right click → *Rename*).

Can't add lights in *Light* tab, they green plus sign is greyed out

» If you want to add more lights to a blueprint, it can't be done here; they have to be added through *Blueprint* tab.

» If you want to add lights to the scene, first select a light that is already in the scene to make sure that the currently active one isn't part of a blueprint.

Can't change the position/direction of a PS/light

» The position (*Link. pos.*) and direction (*Orientation*) of blueprint PSs and lights are affected by the values given in *Blueprint* tab.

  » Note that changing the orientation of a main PS can have an effect on its children.

» If the PS is a part of a weapon, its direction is linked to that weapon.

  » If you want a weapon's child PS move upwards, you can instead use *Vector gravity* under *Particle update*: check *Apply gravity* and *Use scene gravity* and give a negative value for *Scene gravity ratio*.

Changes made to a blueprint's particle system reset

» Make sure you've unchecked *Allow override this PS* under *Blueprint* tab's *Particles* panel for the PS you've edited.

» Make sure that you remembered to copy and save the changes from an assembly to the blueprint.

» Changes made to a blueprint aren't immediately reflected in existing assemblies. You either have to add a new temporary assembly (move to *Blueprint* tab, click somewhere in the scene and press *Insert*) or reset the scene (*F9,* **make sure you have copied the assembly's PS before doing this!**) to see the changes.

Light doesn't blink/isn't affected by the value set for *Cone angle*

» Make sure you have *Blinking light*/*Spot light* checked.

Troubleshooting

Particles aren't boomed/don't bounce/size doesn't change etc.
» There are a lot of options that only take effect if the correct checkboxes are checked/unchecked. For example, if you want particles to boom, check *Also boom particles* under *Emitting* panel; if you want them to bounce, uncheck *Disappear after first bounce* in *Bounce*; if an update related effect doesn't work, check that the corresponding checkbox is checked (*Do sizing*, *Apply gravity*, etc.)

Recoil doesn't do anything
» Besides just the *Recoil* value, the effect also takes into account *Start speed* and *Weight*; if one of these is zero, recoil doesn't do anything.

Replaced the wrong PS in blueprint
» Undo the changes and try again. Make sure that you have the correct PS selected in *Child PSs tree*.

Some of the PSs disappeared after pasting
» Make sure you pasted on top of the correct PS.
» If you're replacing a PS with children with one that doesn't have any/with the paste button with the letters *NC*, the children will disappear.

The blueprint loaded to *Blueprint editor* doesn't show up in *Blueprint* tab
» Try moving to game mode (*F2*) and back.
» Click somewhere in the scene and press *Insert*.
» Try reloading it to editor through *Blueprint library* (*F6*).

Weird (shaky) objects in the scene when editing particles
» These seem to be the remnants of particle trails. If you uncheck *Emit particle trail* in *Particle system* tab's *Trail* panel, the engine still might draw them on screen. You can ignore them.

Sharing your content

An integral part of *Planetoid Pioneers* is the ease of sharing your content with others.

Blueprints, scenes, and particle systems are all saved as PNG files. To access them, locate the *Planetoid Pioneers* folder on your computer. For blueprints, go to *Blueprints* folder (.../*planetoidpioneers/Blueprints*), for scenes *Scenes* folder (.../*planetoidpioneers/Scenes*) and for particles *Particle Systems* folder (.../*planetoidpioneers/Particle Systems*).

There are, of course, many ways to share files, but for the "traditional" approach of copying your assets on a memory stick and giving that forward:

» Once you've found the correct folder, locate the asset you want to share

» On top of it, right-click and choose *Copy*

» Open the folder on the memory stick where you want it pasted

» In that folder, right-click and choose *Paste*

» Eject the memory stick and insert it in the target computer

» There, find the asset file stored on the stick, right-click, and *Copy*

» Locate the fitting *Planetoid Pioneers* sub folder (*Blueprints* or *Scenes* or *Particle Systems*)

» In that folder, right-click and *Paste*

    » If the folder already has a file with the same name, choose *Cancel*/*Skip this file*/ equivalent, rename (right-click → *Rename*) the asset file on the stick and try again

» The asset should now be a part of the target computer's *Planetoid Pioneers*' *Blueprint*/*Scene*/ *Particle system library* and can be accessed through the editor like any other asset.

Sources

This guide didn't come from nothing, existing tutorials were used to help build a comprehensive image of the editor's properties.

"**Adding a Hat to your Pioneer with Joints**" by Steam user MKSTER
https://steamcommunity.com/sharedfiles/filedetails/?id=647825796

"**Creating a new Gun with a Particle System**" by Steam user MKSTER
https://steamcommunity.com/sharedfiles/filedetails/?id=655004630

"**Crush2D editors overview"** by Twitch streamer Data01
https://www.twitch.tv/videos/40779594

"**Data building caves in Planetoid Pioneers**" by Twitch streamer Data01
https://www.twitch.tv/videos/8234724

"**Planetoid Pioneers Alpha - Quick Blueprint creation overview**" by Youtuber Sose Cherofsky
https://www.youtube.com/watch?v=BWcReG_e6Fc&list=PLoUxTACijnl6AbOMTG_eYTq7GUo1iIBNA&index=3&t=0s

"**Putting together a Blueprint with Joints**" by Twitch streamer Data01
https://www.twitch.tv/videos/44086906

"**Using Targets, Triggers and Reactors**" by Steam user LemonCrow
https://steamcommunity.com/sharedfiles/filedetails/?id=1224506699

"**Using the Particle System Library**" by Steam user LemonCrow
https://steamcommunity.com/sharedfiles/filedetails/?id=1224461873