THE MAKING OF SHADOW TACTICS

Bereits seit Dezember 2016 sorgen Mimimi Productions' Klingen des Shogun für unzählige virtuelle Todesfälle im feudalen Japan der Edo-Periode. Entsprechend überfällig ist der Blick hinter die Kulissen des international erfolgreichen Überraschungshits.

»Ey, Commandos war so geil, warum hat das eigentlich noch nie jemand mit Ninjas gemacht?«

Noch vor etwas mehr als einem Jahr war das Spiel des Münchner Studios Mimimi Productions den wenigsten ein Begriff, doch dieser Zustand sollte sich rapide ändern. Im Dezember 2016 erschien die Hommage an den zwei Jahrzehnte alten Echtzeit-Strategie-Hit »Commandos« für den PC und strich neben zahllosen Awards reihenweise Traumwertungen ein. Das gleiche Spiel wiederholte sich mit der Veröffentlichung der Konsolenversion im Sommer 2017 – die Rede ist natürlich von »Shadow Tactics: Blades of the Shogun«.

Nächster Halt: Colortown

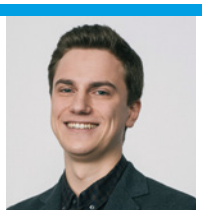
Während die Welt noch schwärmt, sitzt das Team um die Mimimi-Gründer Johannes Roth und Dominik Abé bereits hoch konzentriert am nächsten Projekt. Dennoch fand sich irgendwie noch Zeit, ein »Making of« der Superlative aus dem Boden zu stampfen. Und wie könnte man ein solches besser beginnen als mit einer kurzen Einführung in die Geschichte des noch jungen, aber erfolgreichen Entwicklerstudios, erzählt von CEO Johannes Roth höchstpersönlich?

Ein elementarer Bestandteil dieser Historie ist der direkte Vorgänger des Ninja-Abenteuers: »The Last Tinker: City of Colors«. Auch wenn diese zwei Titel nicht unterschiedlicher sein könnten, so spielen doch beide eine Schlüsselrolle im Werdegang des inzwischen international bekannten Studios.

Der Prototyp des kunterbunten Rätselspaßes nämlich entstand bereits während der Studienzeit, als das Vollzeitprojekt Shadow Tactics noch lange nicht zur Debatte stand. Qualitativ hochwertig sollte dieser Prototyp sein, um dem auch heute noch bestehenden Kernteam bei Vorstellungsgesprächen einen ordentlichen Schub zu verpassen – von einem vollwertigen Titel war damals aber noch nicht die Rede.

2012 schließlich gründete das Sextett die Firma Mimimi Productions, um ihre Mobile App »daWindci« zu veröffentlichen. Dem folgte die Weiterentwicklung von The Last Tinker, die durch finanzielle Unterstützung seitens des FFF Bayern (FilmFernsehFonds Bayern) sowie durch einen Vertrag mit Ravensburger Digital ermöglicht wurde. Regelmäßiges Feedback von Events wie der gamescom flossen in die Arbeit des kunterbunten Plattformers ein, bevor sich mit Unity schließlich ein Publisher fand.

»Wir hatten ein Jahr Zeit, um The Last Tinker fertigzustellen. Hätten wir das nicht geschafft, hätten wir uns überlegen müssen, ob wir in der Branche nicht doch erst einmal anderweitig Erfahrung sammeln«, erinnert sich Johannes. Der Plan ging jedoch auf und nach einem vollen Jahr Entwicklungsarbeit folgte im Mai der Release des farbenfrohen Abenteuers.



Johannes Roth
ist Mitgründer und Geschäftsführer von Mimimi Productions.

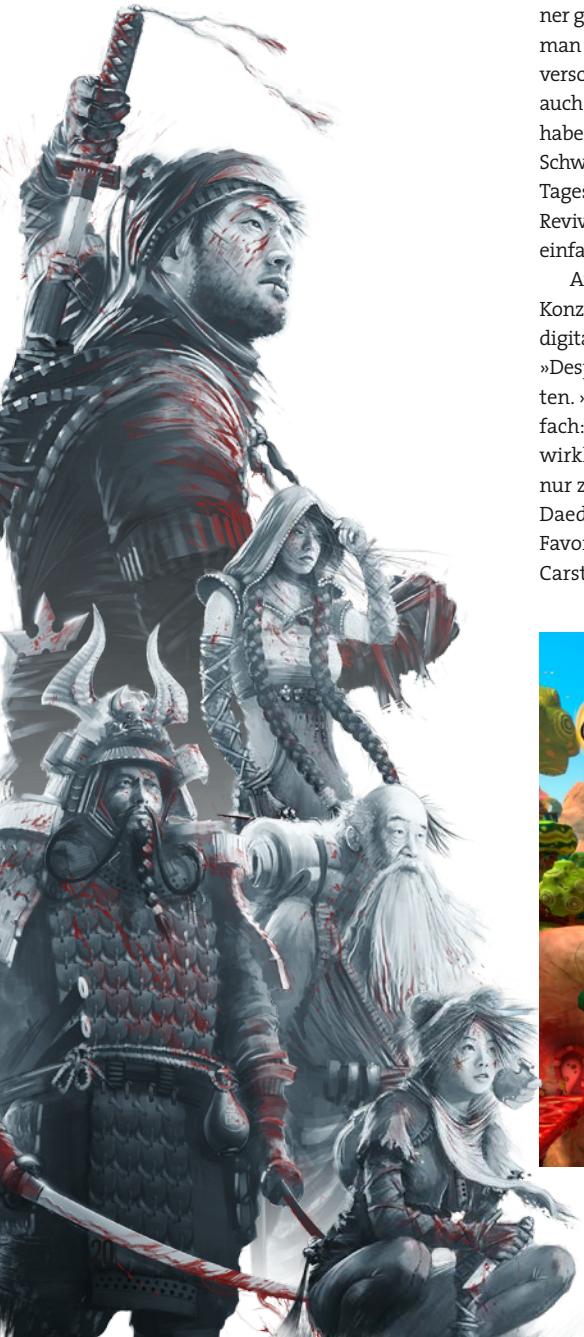
Bereits im Jahr 2008 scharte Johannes das noch heute existierende Kernteam von Mimimi Productions zusammen und gründete zusammen mit Creative Director Dominik Abé das erfolgreiche und kontinuierlich wachsende Studio, das für seine Spiele fortlaufend mit Awards ausgezeichnet wurde; für Shadow Tactics gab es auf Metacritic sogar das für ein deutsches Spiel beste Rating seit 2011. Darüber hinaus steht Johannes Studios weltweit beratend zur Seite, ist Vorsitzender der Games/Bavaria Munich sowie Vorstandsmitglied des GAME Bundesverbandes und stets darum bemüht, die Welt für Entwickler ein klein weniger besser zu machen. @darkrage64

Prototyp, Pitch & Publisher

»Nachdem wir The Last Tinker veröffentlicht hatten, mussten wir uns überlegen, was wir als Nächstes angehen wollen«, erzählt Johannes. »Wir hatten vor Tinker sogar einige Pitches, einer davon war ‚Commandos mit Ninjas'. Die Idee hatte Dom bereits im 1. Semester, da hat er zu Mo gesagt: ‚Ey, Commandos war so geil, warum hat das eigentlich nie jemand mit Ninjas gemacht? Das wär doch perfekt: Du gehst mit einer Handvoll Ninjas gegen eine Übermacht vor, die du heimlich, still und leise Ziel für Ziel ausschaltest, bis am Ende keiner so richtig weiß, was da eigentlich gerade passiert ist!'. 2014 war auch die Zeit, in der Firaxis ihr »XCOM«-Remake veröffentlichten und damit sogar auf Mobile erfolgreich waren. Anfangs haben wir daher sogar darüber nachgedacht, Shadow Tactics für Mobile zu entwickeln, haben uns dann aber für PC und Konsolen entschieden. Wir haben anschließend einen Prototyp in Form eines Mini-Levels entwickelt, in dem man zumindest schon herumlaufen und zwischen Tag und Nacht wechseln konnte. Auch die Sichtkegel der Gegner gab es bereits und wurde man entdeckt, ist man gestorben. Diesen Prototyp haben wir an verschiedene Publisher geschickt. Wir hatten auch andere starke Ideen, aber an diesen Pitch habe ich damals besonders fest geglaubt. Schwere Hardcore-Titel, wie sie früher an der Tagesordnung waren, erlebten gerade ihr Revival, und zu diesem Zeitpunkt hat es sich einfach richtig angefühlt.«

An den Publishern hingegen ging das Konzept derweilen vorbei, obwohl sich die digitalen Fassungen von Commandos und »Desperados« gerade erst wieder gut verkauften. »Der Pitch war eigentlich supereinfach: ‚Commandos mit Ninjas' – das sagt ja wirklich alles. Interesse hatten am Ende aber nur zwei, drei Partner, und einer davon war Daedalic, die sich letztlich auch als unser Favorit herauskristallisierten. Wenn sich Carsten [Fichtelmann], Poki und das Team sich einen Titel raussuchen, weiß man, dass man nicht ein Partner unter Hunderten ist. Daedalic sind selbst Entwickler, entsprechend sensationell ist das Commitment, und dank der Bastei-Lübbe-Geschichte hatten Daedalic damals gerade selbst Geld, sonst wäre der Deal gar nicht möglich gewesen. Wir haben unterschrieben und daraufhin Shadow Tactics entwickelt.

Insgesamt haben wir rund 21 Monate an der PC-Version gebastelt und die Zeit ab dem PC-Release im Dezember 2016 in den Konsolenport gesteckt. Ursprünglich geplant war eine Entwicklungszeit von 18 Monaten und während der ersten 12 Monate lagen wir auch noch perfekt im Zeitplan. Die Sache war die: Nebenher haben wir immer an unserem Vertical Slice gearbeitet – dieses eine Level, das immer polished war und als Qualitätsreferenz diente, während der Rest eher funktional war. Der Vertical Slice war also immer besonders hübsch – nach der Alpha haben wir dann festgestellt, dass das Ergebnis eigentlich deutlich schöner ausgefallen ist, als wir es uns anfangs vorgestellt hatten. Wir haben in diesen 12 Monaten unwahrscheinlich viel gelernt, sei es über unsere Tools, über das Spiel selbst oder auch wie die Grafik am besten wirkt. Und uns war klar, dass wir den Rest des Spiels nun eigentlich in exakt dieser Qualität durchziehen mussten – dafür war unser Zeitplan aber natürlich nicht ausgelegt! An diesem Punkt haben wir begonnen, auch die finanziellen Rücklagen von Mimimi Productions ins Projekt zu stecken. Am Ende des Jahres war's schon recht hart, als einerseits noch alle möglichen potenziellen Folgeprojekte kurzfristig weggebrochen sind, andererseits viele potenzielle Partner erst einmal abwarten wollten, wie sich Shadow Tactics denn so schlägt, bevor sie mit uns einen Deal abschlossen. Und auch das Team war supergespannt, immerhin hat der Titel uns allen unbeschreiblich viel bedeutet. Wir waren gespannt auf die Ratings, Community-Bewertungen, Verkäufe – einfach alles.«



Von Kritikern geliebt

Shadow Tactics erschien am 6. Dezember 2016, einen Tag später fand die Verleihung des Deutschen Entwicklerpreises statt, an dessen Abend Mimimi Productions in satten vier Kategorien das Siegertreppchen erklommen. Gleichzeitig trudelten die ersten Kritiken ein. »Die Reviews waren sensationell: eine 86er-Wertung von der GameStar, 92 Punkte von PC Gamer und sogar Rock Paper Shotgun und Kotaku haben uns mit Lob eingedeckt. Ein solches Ergebnis konnten wir nach 12 Monaten Entwicklungszeit aber natürlich nicht absehen. Als wir damals im Kollektiv die Entscheidung getroffen haben, das Spiel in der höchstmöglichen und nicht der ursprünglich geplanten Qualität fertigzuentwickeln, habe ich klar und deutlich gesagt, dass wir mit solch einer Entscheidung 'all in' gehen würden – im Falle eines Misserfolgs wäre es das dann auch ziemlich sicher gewesen. Die einstimmige Antwort lautete: ‚Dann müssen wir's eben so geil wie möglich machen, immerhin wollen wir ja auch das Genre wiederbeleben!'

Am Ende kam es, wie angekündigt – zum Zeitpunkt des DEP waren wir faktisch pleite. Trotzdem war das ganze Team vor Ort, sogar unsere Praktikanten, die wir zwischenzeitlich bei uns hatten, waren mit von der Partie, und jeder hatte seine Bahntickets selbst bezahlt. Wir wollten einfach einen geilen Abend haben und zusammen feiern, hatten aber alle im Hinterkopf, dass es am 31.12.2016 aus ist und eigentlich nur noch diejenigen aus dem Team da wären, die mit der Konsolenfassung beschäftigt waren.

Nun ja, wir hatten hoch gepokert … und die Wette glücklicherweise gewonnen. Das Spiel kam super an, und ab diesem Zeitpunkt gestaltete sich auch die Suche nach weiteren Verhandlungspartnern deutlich einfacher als zuvor. Und ich muss einfach positiv hervorheben, dass wirklich niemand versucht hat, unsere damalige Situation auszunutzen und uns mit einem schlechten Deal über den Tisch zu ziehen. Dazu kommt, dass sich Verhandlungen mitunter ziemlich in die Länge ziehen können, und auch das war glücklicherweise nicht der Fall.

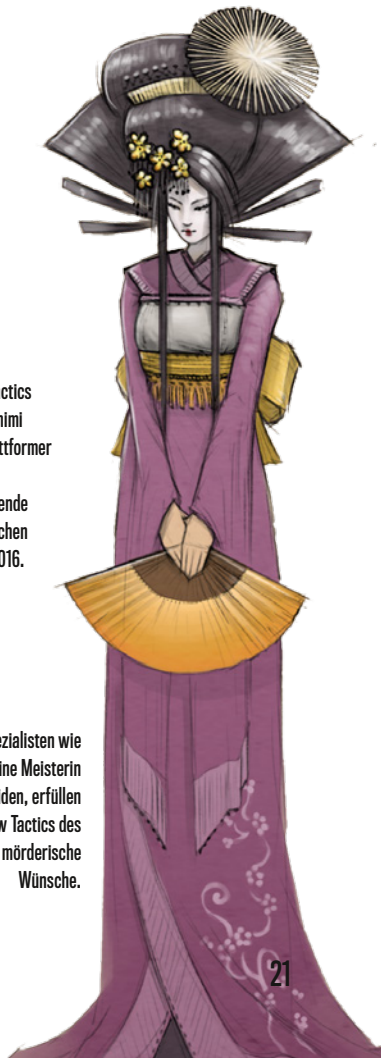
Ich weiß noch, wie ich am 23. Dezember im Zug saß – zu dem Zeitpunkt waren alle schon unterwegs in den Urlaub – und Mails an das Team geschrieben habe, um mitzuteilen, dass der Vertrag unterzeichnet und wir erst mal sicher sind. Anfang Januar sind wir dann zusammengekommen und haben feierlich sämtlichen Papierkram vernichtet, der irgendwie mit dem potenziellen Untergang unseres Studios zu tun hatte (lacht).«

Fakten, Fakten, Fakten

»Natürlich haben dir die ganze Zeit auch über darüber sinniert, wie sich Shadow Tactics denn zahlenmäßig machen wird und wie viele Einheiten wir effektiv an den Spieler bringen. Sollten wir im Dezember etwa 10.000 Stück verkaufen – so meine persönliche Rechnung –, dann dürfte sich das ganze Projekt so in ein, zwei Jahren langsam rechnen, das Investment wäre nach und nach eingespielt und die ganze Aktion zumindest keine finanzielle Katastrophe. Ab 30.000 verkauften Einheiten wären wir richtig gut dabei, 50.000 hingegen wären ein echter Traum – letztlich gingen 60.000 Stück über den Ladentisch, gegen Ende Januar waren es dann sogar schon 100.000. Aktuell liegen wir, Konsolen nicht inbegriffen, bei rund 200.000 verkauften Einheiten über Steam, GOG und diverse Plattformen in China – ein für uns überraschend starker Markt! Erwähnenswert ist im Zusammenhang mit diesen Zahlen, dass wir bisher nie Rabatte über 33 Prozent gegeben haben!«

Ob denn inzwischen Geld hängenbleibe? »Bereits seit Januar machen wir Gewinn«, berichtet Johannes stolz. »Reich sind wir noch lange nicht und von Luxus fehlt weiterhin jede Spur, wie die Ikea-Möbel im Studio beweisen, aber wir können regelmäßige Gehälter zahlen. Unsere Arbeit am aktuellen Projekt

Linke Seite: Vor der Arbeit an Shadow Tactics veröffentlichten Mimimi Productions den Plattformer The Last Tinker.
Rechte Seite: Strahlende Gewinner des Deutschen Entwicklerpreises 2016.



Fünf Spezialisten wie Aiko, eine Meisterin im Verkleiden, erfüllen in Shadow Tactics des Shoguns mörderische Wünsche.

finanzieren wir derweil über die Mittel, die uns unser neuer Publisher zur Verfügung stellt.

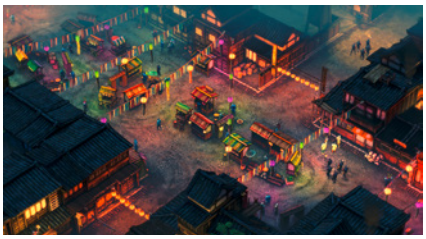
Für das Geld, das wir derweil mit Shadow Tactics verdienen, gäbe es zwei mögliche Verwendungszwecke: Wachstum oder Rücklagen. Wir haben uns für Letzteres entschieden, denn eine Situation wie vergangenen Dezember möchte ich dem Team nie wieder zumuten müssen. Es ist unglaublich besch&§%en, Leuten mitteilen zu müssen, dass sie bald ziemlich sicher keinen Job mehr haben werden. Klar, am Anfang sieht man das noch etwas lockerer, hat noch Hoffnung, bis es schließlich immer ernster wird. Mich persönlich hat eigentlich nur beruhigt, dass wir immer die besten Leute im Team hatten und jeder mit Sicherheit irgendwo unterkommen würde. Wir hatten inzwischen einen Namen und das Spiel war top, in diesem Sinne waren wir gut aufgestellt. Aber auch wenn sich gegen Ende Dezember alles langsam in Wohlgefallen auflöste, hab ich es lange nicht geschafft, richtig abzuschalten. Der ganze Umstand hat mich unterbewusst noch über die nächsten sechs Monate belastet und sich in Unruhe und Schlafstörungen manifestiert. Auch finanzielle Stunts à la Gehälter aussetzen kamen zu keiner Zeit in Frage. Unterm Strich macht man damit auch nur Schulden, und das bei Leuten, die ihr Geld schlicht und ergreifend brauchen. Private Rücklagen hatte niemand, wir kamen ja direkt aus dem Studium und hatten teils sogar noch ein paar saftige Kredite am Hals, die indirekt ja auch unser Unternehmen mit abbezahlt.

Kontrolliertes Wachstum

Gewachsen sind wir zwar, aber lediglich auf 25 Personen, von denen 23 hier sitzen und zwei freie Mitarbeiter sind. Dabei soll es erst mal bleiben, das Studio ist jetzt eh voll. Und ich bin ehrlich gesagt froh drum, weil es sonst doch wieder darauf hinausläuft, dass man hier noch einen coolen Praktikanten findet und da noch jemanden unterstützen möchte und so weiter. Wir haben uns vielmehr darauf konzentriert, diverse Prozesse zu verbessern. Tobias [Filthaus] haben wir an Bord geholt, nachdem er in der Closed Beta auf Steam unglaublich aktiv war. Wir haben ihn gefragt, ob er Bock auf QA hätte – tja, jetzt ist er fester Bestandteil des Teams und kümmert sich direkt von Beginn um die QA in unseren Titeln. Tom [Kersten] kam nach fünf Jahren von Daedalic zu uns und kümmert sich als Production Director um das Projektmanagement, was wiederum Dom und mich in Sachen Planung entlastet. Und dann haben wir das Team noch um unseren UI Artist Reinier [Goijvaerts] sowie einen zusätzlichen Level Designer erweitert. Wir haben tolle Artists im Team, UI Design ist aber nochmal eine ganz andere Nummer und war schlicht und ergreifend nicht deren Kernstärke; einen Spezialisten dafür im Haus zu haben, der konkret mit den Concept Artists kooperiert, ist einfach spitze. Abgesehen von den Neuzugängen konnten wir zudem Leute übernehmen, deren Verträge noch befristet waren.

Wir haben letztlich keine Fluktuation im Haus, und das ist mir persönlich unglaublich wichtig. Ich frage Bewerber in Vorstellungsgesprächen auch ganz direkt, ob sie es sich vorstellen können, zehn Jahre in derselben Firma zu bleiben. Mir ist schon klar, dass man so eine Frage nicht wirklich sinnvoll beantworten kann. Es geht mir dabei aber auch eher darum, Leute dazu zu bringen, sich mit diesem Gedanken überhaupt einmal auseinanderzusetzen.«

Die Assassinen des Shogun werden über 13 Missionen hinweg mit sämtlichen Witterungen und Tageszeiten konfrontiert - einfach kann schließlich jeder.



Keine Fluktuation zu haben, ist auf zweierlei Arten ein positives Zeichen: Zum einen geht es der Firma gut genug, um niemanden gehen lassen zu müssen. Zum anderen spricht es für das Arbeitsklima in einem Betrieb, wenn der Wunsch nach einem Wechsel gar nicht erst aufkommt. »Das ist eigentlich echt verrückt …«, schießt es Johannes plötzlich in den Kopf. »Wir, also zumindest das ursprüngliche sechsköpfige Kernteam, feiert 2018 sein 10. Jubiläum. Seit 2008 machen wir in diesem Team schon Spiele, von der tatsächlichen Firmengründung jetzt einmal abgesehen. Das hat damals angefangen mit »Grounded«, als wir uns in den Weihnachtsferien noch gegenseitig Levelskizzen zugeschickt haben … das ist echt schon lange her …

Wie dem auch sei, aktuell geht's uns gut – wir sind nicht reich, schwimmen nicht in Geld und legen das, was wir haben, zur Seite. Wir haben einen tollen Partner und ein super Team. Und es ist ok, einfach mal Steuern zu zahlen, anstatt auf Teufel komm raus zu investieren und zu wachsen und am Ende wieder in eine Situation zu kommen, in der Leute unsicher sind. Wir wollen uns auch weiterhin immer auf ein Projekt konzentrieren, anstatt mehrere gleichzeitig am Laufen zu haben, das ist einfach nicht unser Ding. Wir legen großen Wert darauf, dass unsere Leute ihre Kernkompetenzen auch voll ausfahren können. Gibt es mehrere aktive Projekte, muss man zwangsweise umschichten und je nach Bedarf Team-Member aufgrund ihrer Kompetenzen dann an Stellen setzen, obwohl sie andernorts besser aufgehoben wären.«

Neue Projekte statt DLC

Dass diese Einstellung Fluch und Segen zugleich sein kann, wird klar, als Johannes auf die Frage nach neuen Projekten und möglichem DLC zu Shadow Tactics eingeht. »Wir arbeiten bereits an unserem neuen Projekt. Ich kann leider noch nichts Konkretes sagen, aber wir bleiben dem Genre treue – ein offizieller Nachfolger zu Shadow Tactics wird es aber nicht werden. Das mit DLC für Shadow Tactics ist wiederum eine andere Geschichte … wir hätten unglaublich gerne zusätzlichen Inhalt entwickelt und hatten auch echt coole Ideen, aber diesbezüglich mussten wir einfach realistisch bleiben. Der Punkt ist ganz einfach der: Ein Level ist ein Level und der Arbeitsaufwand für eine DLC-Map exakt der gleiche wie für eine Map im Hauptspiel und an beide stellen Spieler die gleichen Qualitätsanforderungen. Das bedeutet im Umkehrschluss, dass beinahe das gesamte Team involviert sein muss, denn es braucht die besten Leute, um diese Qualität erneut zu garantieren. Gleichzeitig aber brauchen wir diese Leute eigentlich schon für das neue Projekt, das sich dann zwangsweise nach hinten verlagern würde. Dazu kommen Lokalisierungen und Vertonungen, denn sämtlicher DLC muss logischerweise ins Japanische übersetzt werden, und das wird, wenn es nicht in einem Rutsch erledigt wird, schnell teuer – so ein DLC muss sich am Ende ja auch rechnen. Der finanzielle Aspekt spielt ganz klar eine Rolle, noch wichtiger ist aber dennoch der zeitliche. Neue Leute dafür anheuern? Die müssten sich erst einmal komplett in das Projekt einarbeiten, wobei sie wiederum die Hilfe des Hauptteams bräuchten, das dafür aber eigentlich keine Zeit und möglicherweise sogar schon neue Technologien entwickelt hat, die dem DLC-Team unbekannt sind – eine endlose Geschichte. Der langen Rede kurzer Sinn: Wir haben uns dafür entschieden, unsere Kapazitäten und unser Herzblut stets zu 100 Prozent in ein einziges Projekt zu stecken. Wir hätten gerne DLC entwickelt, das neue Hauptprojekt hat aber ganz eindeutig Vorrang und würde nur darunter leiden. Und einen günstigen DLC mit minderer Qualität zu veröffentlichen, nur um Geld zu machen, stand für uns nie zur Diskussion.

»Einen günstigen DLC mit minderer Qualität zu veröffentlichen, nur um Geld zu machen, stand für uns nie zur Diskussion.«

Damit keine Verwirrung aufkommt: Mit der Arbeit am neuen Projekt haben wir bereits im Januar begonnen, also noch während am Konsolenport gemeiselt wurde. Dabei handelte es sich aber um rein technische Angelegenheiten, neue Inhalte wurden ab diesem Zeitpunkt nicht mehr entwickelt. Wir wussten von Anfang an, dass wir eine Konsolenfassung haben wollen, dementsprechend haben wir von Tag eins an diverse Aspekte beachtet. Eine Controller-Steuerung haben wir bereits für die PC-Fassung entwickelt und dabei darauf geachtet, beispielsweise alle Skills unterzubringen. Es stand nie zur Debatte, unterschiedliche oder gar abgespeckte Fassungen zu entwickeln. Darüber hinaus haben wir regelmäßig technische Features gegengeprüft,

Portierung & Lokalisierung

Neben einer englischen Sprachfassung entschied sich das Team nicht für eine deutsche, sondern eine japanische Lokalisierung. Gemessen an Thema und Setting von Shadow Tactics äußerst passend, aber dennoch ungewöhnlich.

»Shadow Tactics eine japanische Lokalisierung zu spendieren, hatte nichts damit zu tun, um in Japan mehr Erfolg zu haben«, erläutert Johannes. »Aktuell ist das Spiel dort sogar nur für den PC erhältlich und der genießt in Japan bekanntlich keinen allzu großen Stellenwert. An der Konsolenfassung arbeiten wir derzeit, es gibt allerdings noch einige marktspezifische Eigenheiten auszuloten. Der Entschluss rührt einfach daher, dass wir die Idee megageil fanden (lacht) – das Plus an Flair ist unglaublich, und das hat uns die Community auch gedankt.

Klar, in Deutschland hagelte es dafür von diversen Seiten Watschen, dass es keine deutsche Übersetzung ins Spiel geschafft hat. Fakt ist aber nun einmal: Wir mussten uns rein finanziell für eine von beiden entscheiden – die japanische oder deutsche. Tja, das Ergebnis ist bekannt. Und auch an dieser Stelle hatten wir mit Daedalic als Partner unglaubliches Glück, da sie unsere Motivation hinter diesem Schritt von vornherein verstanden haben.«

Sehr subtil: Explosionen dienen als gezielte Ablenkungsmanöver, um die Aufmerksamkeit der Wachen in ein bestimmte Richtung zu lenken.

Oben: Die polnische Collector's Edition von Techland entspricht in kleinerem Format der deutschen, auf 1.000 Stück limierten und bereits vergriffenen Fassung.
Unten: Co-Founder und Creative Director Dominik Abé präsentiert stolz eines der Goodies der Sammleredition.

etwa ob wir eine vernünftige Quick-Save-Funktion auch auf der Konsole zum Laufen bekommen. Ansonsten haben wir uns aber zunächst rein auf den PC konzentriert und uns erst im Anschluss dann zu 100 Prozent um den Port gekümmert.

Deutlich anspruchsvoller war es, mit den technischen Gegebenheiten der Konsolen dieselbe Performance zu erreichen – wir ziehen alle unsere Hüte vor Entwicklern wie Naughty Dog, wenn man sieht, was die aus einem »Uncharted« herausholen. Fakt ist nun mal, dass die Prozessoren in den Konsolen bei Weitem nicht dem entsprechen, was in aktuellen Mittelklasse-Rechnern zu finden ist. Klar, Shadow Tactics sieht vielleicht nicht aus wie Uncharted, dennoch hat jeder Level über 100 Gegner und die wollen alle gleichzeitig berechnet werden. Dazu hier und da die Charaktere platzieren, während irgendwo am anderen Ende der Karte eine Patrouille ihre Runden dreht. Dabei will zu jeder Zeit die Route, die KI, der Sichtkegel und natürlich das berechnet werden, was der NPC sieht und hört – ganz egal, ob sich der Spieler in diesem Bereich oder ganz wo anders aufhält.«

Lehren und Zukunftspläne

Ein Projekt wie Shadow Tactics zu stemmen, passiert nicht über Nacht, und unabhängig vom tatsächlichen Erfolg ist das Team vor allem hinsichtlich einer Sache deutlich reicher: Erfahrung. Ob sich die denn auch direkt auf das Folgeprojekt auswirkt?

»Im Gegensatz zu The Last Tinker haben wir bei Shadow Tactics recht bald sehr genau gewusst, wen unsere Zielgruppe umfasst. The Last Tinker war zwar bunt und kindlich, aber dennoch fordernd, und irgendwie hat das nicht so gut zusammengepasst. Shadow Tactics war von Anfang an darauf ausgelegt, schwer zu sein, und dadurch haben wir einen ganz spezifischen Kreis an Spielern angezogen. Das war über die gesamte Entwicklungszeit hinweg eine super Erfahrung und es war ein gutes Gefühl, eine gewisse Sicherheit zu haben.

Eine der größten Hürden, vor der wir standen, war mal wieder die Vermarktung. Klar, man kann sich bei SteamSpy anmelden und nachschauen, wie viele Leute sich Commandos gekauft und wie viele davon sich auch für Shadow Tactics entschieden haben. Es bleibt dennoch unglaublich schwer, die Leute da draußen zu erreichen – trotz sämtlicher geiler Ratings und Berichterstattung. Erst vor kurzem hat mich jemand darauf angesprochen, dass er unser Spiel toll findet, aber dass er erst zwei Wochen vorher davon gehört hätte und unser Marketing echt mies wäre. Und ich dachte mir nur: ‚Sorry, wir waren wirklich auf sämtlichen Kanälen vertreten, auf sämtlichen Frontpages und in allen Printmedien und vor kurzem sogar im TV zu sehen – was sollen wir denn noch machen?' Wir kamen sogar bei ein paar großen Streamern und Let's Playern unter, und nicht einmal das hatte derjenige gesehen. An der Bushaltestelle Plakate aufzuhängen können wir uns dann eben doch nicht leisten. Es ist nach wie vor sehr schwer, sowohl die alten Commandos-Fans abzuholen als auch jüngere Spieler zu begeistern, die die alten Titel gar nicht kennen. Es ist eben kein cineastisches Blockbuster-Spiel und entsprechend schwer zu bewerben. Wenn die Leute aber erst einmal selbst Hand anlegen, beispielsweise auf Messen, dann finden es alle total geil und wir hören Sätze wie ‚Das ist ja endlich mal ein vernünftiges »Assassin's Creed« in schwer und mit mehreren Charakteren' oder ‚Stark, wie »Metal Gear« in Iso-Perspektive!'

»Es ist nach wie vor sehr schwer, sowohl alte Commandos-Fans abzuholen als auch jüngere Spieler zu begeistern.«

Den Leuten das nahezubringen, wird auch in Zukunft nicht leichter, aber immerhin kennen wir jetzt schon einmal unsere Zielgruppe und Community und die wird sich wohl auch mit dem nächsten Titel nicht ändern. Glücklicherweise konnten wir mit Shadow Tactics eine Nische besetzen, selbst wenn es uns in Zukunft nun vielleicht andere Studios gleichtun sollten. Sicher fühlen wir uns nicht, und auch der Druck, ein geiles Produkt abzuliefern wird in Zukunft keinesfalls geringer sein – an unserem Ziel genau das zu tun, wird das aber nichts ändern!«

Wünsche für die Zukunft? »Im Moment bin ich tatsächlich rundum zufrieden. Die Gehälter können gerne nochmal steigen, aber immerhin können wir alle unsere Leute fair und regelmäßig bezahlen. Wir können wachsen. Wir haben alles getan, um Überstunden zu vermeiden und getrost Urlaub machen zu können. Genauso klar ist, dass die Arbeit auch in Zukunft stressig bleiben wird. Mein größter Wunsch ist dennoch tatsächlich, immer einen Weg zu finden, das Team so beisammen zu halten, wie es ist.«

Johannes Roth

FINDING THE PERFECT ART STYLE

From colorful to rather dark and realistic: finding the right art style for Shadow Tactics took time, but it was worth every second.

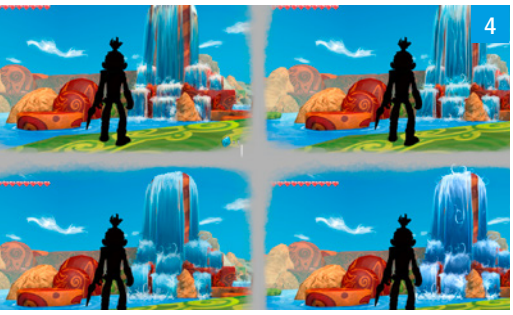
Bianca Dörr
is Art Director and Texture Artist at Mimimi Productions.

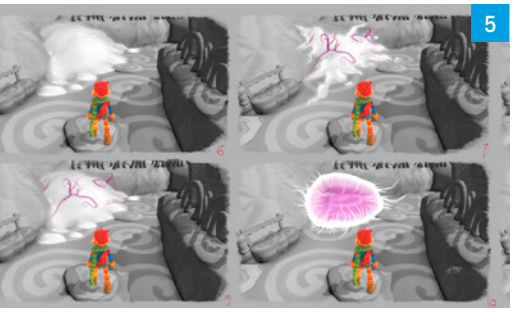
In her position as Art Director, Bianca is with Mimimi Productions since 2012. She's not only in charge of leading and coordinating the art team, but also works on game asset textures and develops the art styles for every major project. On top she is responsible for the quality of all graphics and artworks. @Katzenviechle

Since the company's founding in 2012 we developed two big games for PC and consoles as well as several mobile apps, with these two games being »The Last Tinker« and our recent release »Shadow Tactics: Blades of the Shogun«. In this article we want to take you behind the scenes of our quest to find a fitting art style for both The Last Tinker and Shadow Tactics. First off, we will introduce both titles and their art styles to you. After that we'll dive right into the development process before we talk about problems and challenges we encountered along the journey – and of course about how we solved these issues.

The Art Style of The Last Tinker & Shadow Tactics

The Last Tinker is a third person jump 'n' run in a colorful world, inspired by games like »Banjoo Kazooie« and »Jak & Dexter«. As Koru, you must defeat the Bleakness that threatens Colortown by using the power of colors.





The game's style is defined by round and smooth shapes, bright colors and a warm light setting. Everything is made from paper, color and glue which combined results in papier-mâché. As a contrast, we added some cardboard elements to create edges and make the world look more interesting. No straight lines, no 90 degrees, rather playful than super accurately drawn patterns – everything looks like it's made and painted by children. The game's characters are cartoony and funny, showing their attributes in their looks.

Shadow Tactics is set in Japan's Edo period and it's quite the opposite of The Last Tinker. It's a hardcore realtime stealth game in which you control five different characters with individual skillsets. Basically, it's »Commandos« with Ninjas. You kill or sneak around enemies to accomplish your missions. Theme and story of this game are darker and grittier, that's why we decided to create a more realistic style and combine it with non-realistic elements like outlines imitating ink, that typical Japanese paintings are famous for. We included many details and each map has its own interesting light setup to catch the mood of a level. In comparison to The Last Tinker, we used more desaturated colors. All characters are human but they all have their own unique silhouette, so you can instantly identify and recognize them. However, they have longer legs than normal humans – a design decision we made because the perspective made them look smaller and somehow compressed.

Finding the Style for The Last Tinker

The Last Tinker started as a project while we were still at university and we had three to four weeks to complete it. It had to be in 3D which was a problem for us at that time. When we started developing the style, our artists had nearly no experience in 3D modeling – our strength clearly lied in creating 2D graphics. So we thought about how we could use our skills in creating 2D graphics for a 3D game. We didn't have the time and experience to make high detailed models so we thought about what we could create with the help of simple shapes and if we could create details with the help of textures. That's why we started with a 2D image that we projected on a plane and roughly cut out the shape of the image. Then we extruded the face and voila – we had a 3D model. Now we just needed the right material to realize this style, ending up with cardboard 2.

So we decided to create a world made entirely out of cardboard but we weren't really satisfied with the early outcome. It looked sort of boring. Plus, we felt like cheaters because our 3D models were so simple. Still, we needed 3D models with plain shapes that could quickly be created and animated. Then we had the perfect idea: Papier-mâché! It's perfect for beginners in 3D modeling. Plus, at university we were five artists. After starting the real production, we were two artists, then four. By only having to handle one or two materials we were able to create a world fully made of papier-mâché. Material handling was crucial, as creating different good looking materials that look good and real costs a lot of time. Everybody loved the idea and since we had proper 2D skills, we decided to use textures to add detailed hand-painted patterns.

Now we had a base to work with. To get to know the material better we came together with artists and game designers in tinkering sessions 1, testing different shapes and patterns. We had two or three of these meetings and in the end we really knew the material and its potential. The figures we made were nice references. Some

were really good, some were pretty ugly. Oh, and we still have them in our office. One of them sits right next to me and watches me ... creep ... 3

Anyway, after having decided on a direction, we wanted to get a clearer vision for our design, which we eventually found in the works of Hundertwasser and Niki de Saint Phalle. Both focus on round and smooth shapes which colorful textures. Plus, we looked for papier-mâché figures made by kids because we wanted the world to look like kids built it. Those references gave us a good feeling for how the world and characters should look like.

First Steps in the Right Direction

After graduation, we occasionally experimented with the prototype, got a lot of feedback and wondered how we could improve it. When it became official that we could make The Last Tinker a complete game, we developed a story and reworked the gameplay which both influenced the style of the game. The first thing we noticed was that everything looked too clean and not very lifelike.

So we added some color splashes and dirt to make everything look a bit more used. Along with the story came Colortown's different districts, which served the gameplay well. Prior to that, the world was so colorful and chaotic that players often got lost. Introducing the districts and with them some coloring rules helped solving this issue. As already mentioned, we initially used only two materials. After starting the development, however, we considered adding more materials like water and paper hills 4 to make the world more interesting. We struggled a long time with the water because it never really fit into the world.

Then we had the idea to add patterns that you can see on nearly every asset: spirals. They appear as ripples in the water and as splashes in waterfalls. Honestly, I love spirals and I loved drawing them. After some time though, I was so fed up with them that now, every time I see a spiral, I'm about to scream!

Of course, not all of Tinkerworld is friendly and colorful. The Bleakness wants to erase everything in it. The player sets out on an adventure to defeat the Bleakness and the monsters it created – so much for our vision of the first prototype. Style-wise, they needed to fit into the world, so we gave them round and simple shapes. Still, we wanted them to look like something totally strange and foreign to the world. So we decided to make them out of some kind of goo, however, they seemed like being made out of porcelaine. Again – with the story we added in the later production, in which whole colortwon is covered by bleakness – we needed to find a design for this new material. We had an overall feeling and look for the creatures but we still struggled with the Bleakness that wasn't lifelike.

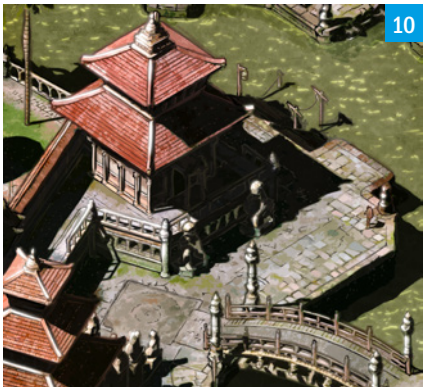
We had so many concepts 5 and it took us very long to get the Bleakness' style right – also because of some technical problems. We ended up with a mix between a less shiny form of goo, and something that you can't touch, something that isn't really quite there. We also added some patterns on every bleak object to give them a little bit of a mystical appearance 6.

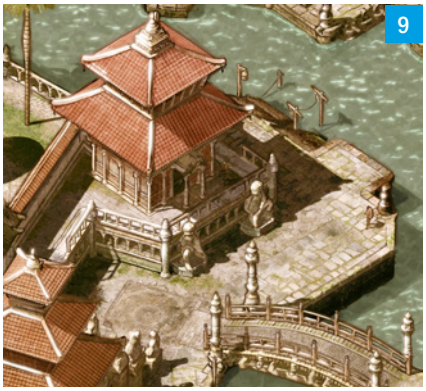
Finding the Style for Shadow Tactics

The basic idea for Shadow Tactics formed during our time at university. Our Creative Director had always been a huge fan of "Commandos", so he thought "Why not revive this genre?" So, what's cool and fits a stealth game? Exactly: ninjas! So we pitched a Commandos-like game with ninjas, with the very first draft being a mobile version. And as we all know, it's always common to reuse skills you acquired in prior projects to

»We wanted to reach the old fans of Commandos which were used to a much more realistic style.«

8 9 10
The concept artists painted over Commandos screenshots in terms of analyzing and finding a fitting art style.





14 While the vertical slice served as quality reference, the rest of the levels looked rather rough.

do something completely different ... not. Shadow Tactics was supposed to become a mobile game 7, so we had a rather cute and cartoony style in mind, which we were already used to. Or in other words: we were – artistically speaking – kind of stuck and pretty much too influenced by our work on The Last Tinker, but we knew that this wasn't the style we wanted to go for. We wanted to reach these old fans of Commandos which were used to a much more realistic style with tons of details and crispy textures.

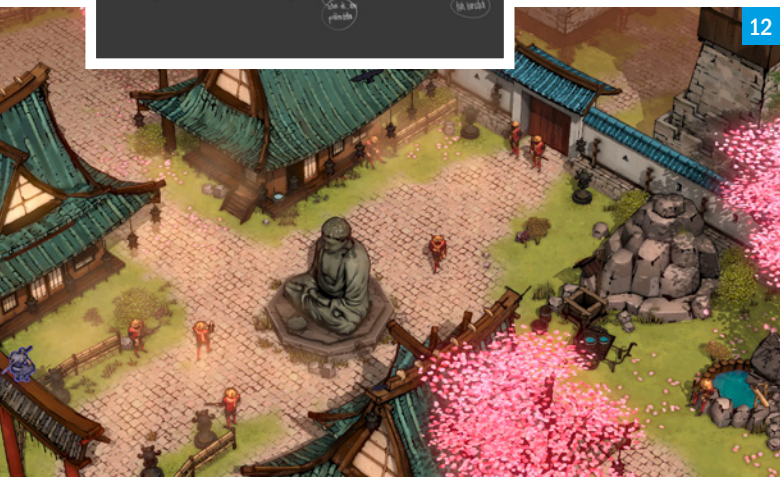
We analyzed the spiritual predecessor 8 and tried to understand what made Commandos ... well, Commandos. We wanted to keep its core elements within in our own style to keep the old fans of the game happy – for us artists, however, finding the right style became a real struggle. We had our concept artist overpaint some Commandos screenshots 9 10 to get a better feeling for a more realistic style while trying to add our touch to it. We never wanted to create some super realistic style, because that just wouldn't have been us. Plus, for only four artists it was nearly impossible to make a realistically looking game, so we mixed realistic elements with non-realistic elements. We started to concept and model some buildings because somehow the concepts we had created so far didn't get us any further. In the beginning, we reduced the complexity of the models and again used textures to obtain detail, which based on photographs that our artists painted over to give them a painted look. We experimented with different non-realistic elements like outlines and canvas patterns 11. The outlines worked best ingame and fit the Asian style the most, while the canvas patterns created too much noise and simply didn't look good. After finishing the first playable prototype 12 13, we already knew that we also wanted to include numerous interesting light-settings for catching different moods.

Shadow Tactics' pre-alpha

For our vertical slice we created polished assets to get a feeling for the style and the quality we wanted to reach, so a lot of time went into one level 14. Between putting up the prototype and this pre-alpha version we had time to get used to the perspective that brought its own challenges with it. We changed the color palette and various little details, because the prototype seemed a bit to cheerful and idyllic. Shadow Tactics' story included wars and tragedies, so we developed a better fitting style, using more desaturated colors, but still keeping it very painty, though with reduced details and thick outlines.

From there on out we received some feedback from friends from other games companies, many of them asking „Oh, is it a mobile game?" Just what you want to hear when you're developing a big PC and console game ... it really bothered us. We realized that our current version was too far away from the original Commandos style, lacking detail, crispiness and realism. We used no normal

11 A collection of different outlines and patterns
12 The prototype level at daytime ...
13 ... and at night



14 Vertical slice
15 Pre-alpha version
16 Pre-beta version

or specular maps which therefore had no depth and small surface details – everything was kind of plane. This particular level admittedly wasn't the best choice for a vertical slice, because it was one of our most colorful ones – not the best basis for finding the right art style for a game that's getting darker and darker in the long run.

From Pre-Alpha to Alpha

When we created our alpha version 15, we added some normal as well as some specular maps for metallic assets, also adding further details to textures like on stones and thatches. We changed the way that nature looked, from a very painty to a more realistic style, with small leaves instead of big spots. We also removed the outlines from all foliage. Additionally we used the new lighting system from Unity 5. However, we first wanted to get the overall setup right, so we lost the interesting lighting at that stage. And still, it wasn't what we were aiming for ...

From Pre-Alpha to Pre-Beta

In our pre-beta version 16 we decreased the overall number of outlines and made them thinner. We even considered deleting them completely but lost that thought because they created a nice depth and contrast between asset and terrain. We also added dirt to the textures to give everything a gritty and used look. And our shaders learned color variations which you can see on the stones. At that time we didn't have a good setup, that's why it looks a bit out of place. Besides making the scenes more realistic, however, it kind of masked the fact that we used the same assets over and over again. We tweaked a lot of small things in this phase of development and improved our terrain by adding normal and height maps. These height maps did a better job in blending textures, so the transition between two materials like grass and dirt looked rather crispy than blurry. Eventually, we gave every level a nice light-setup and used color mapping to create more variety between the levels thus creating the proper mood. Ambient particles like in The Last Tinker made the world more lifelike. Well, so much for finding and creating the right art style for Shadow Tactics.

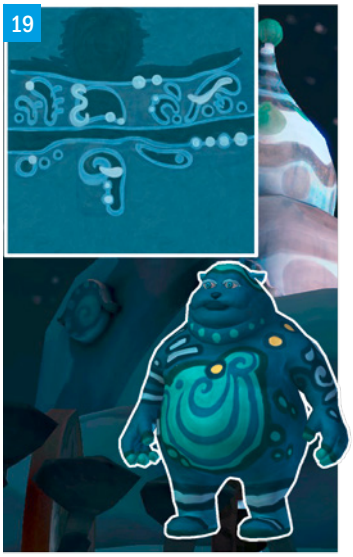
Problems and Solutions

On our quest to find the right art style for both The Last Tinker and Shadow Tactics we certainly had to overcome quite a number of challenges. So, let's take a look at what these problems were and how we managed to solve them.

The Last Tinker

As mentioned earlier, the first version of The Last Tinker was too colorful and chaotic. Players got lost on a regular basis and didn't know where to go. Since we used every color to paint the assets and usually had a bright light-setup, we couldn't use color or light to guide the player through the world. As we developed the story we divided Colortown into different districts. That helped us bringing some order into this colorful world. Bye reducing the number of colors per district and, at the same time, adding specific shapes and patterns as well as some coloring rules, we were able to help players navigate better. Plus, every district is inhabited by different races 17 18 19. The whole game uses the same house assets. So by giving each districts its own attributes and uniquely designed assets, we could create enough variety and saved a lot of time by not modeling different house types.

Another way to facilitate navigation was adding unique landmarks like a big windmill 20 or unique fountains or maybe special buildings like taverns, that draw the players' attention and thereby help them remember certain places. Unfortunately, we never managed to completely solve this issue so eventually we gave our companion Tap a guiding feature. Whenever you call him, he shows you the way before you get lost again. Anyway, we just yellow to mainly



17 18 19 Different colors were used to better organize and define The Last Tinker's different districts.

mark relevant gameplay assets climbable walls, pillars you can jump on or specific landmarks. So, whenever you see something yellow, that's where you want to go!

The Challenges of Creating Shadow Tactics

Shadow Tactics confronted us with a whole set of different problems, the biggest being the style itself. On the one hand we weren't sure on which platform we'd release the game. On the other hand our artists were still too influenced by our previous projects that were all cute and cartoony. »Realism« was another big thing. Not only did we want to make a more realistic looking game, we also wanted it to be historically correct. For that we had to do a lot of research and research takes up a lot of your time! Time – unsurprisingly – was a big problem, but well ... when isn't time a problem game development 21? As mentioned in the beginning we switched between styles a couple of times, until we were told it looked like a mobile game and we started to increase the quality, added normal and spec maps, better lighting and more details overall, which resulted in a ton of additional work. And with us still being only four artists, time was crucial. Also, we lacked experience in many ways, e.g. when it came creating high-detailed terrains and proper lighting setup, so we had to get to know a lot of new tools which – again – cost us a lot of time we didn't have. Oh, did I mention the time problem? Well, luckily we have a lot of comfortable couches in our office.

When we had finally developed the first style-prototypes and started with the creation of assets, we encountered a new problem.

The Perspective

In the beginning we had really large curved roofs that took up a lot of screen space which made it difficult for our level designers to place enemies the way they wanted. It also made it difficult to navigate the player characters because you couldn't see what was behind the buildings. It was possible to rotate the camera but with all these roofs the player wouldn't have been doing much else, which would have eventually killed the fun. Also, the big roofs looked kind of out of place, so we iterated them a couple of times until they fit. We minimized the roof sizes and flattened 23 them so they wouldn't hide the enemies on the ground.

The Architecture

(56.) Another problem was the Asian architecture. We wanted our characters to be able to climb roofs to generate that sneaky ninja feeling. But Asian roofs are rather bent and curved and these shapes collided with our animations. The feet of the characters didn't adapt to the ground beneath them, so it always seemed like they were floating. Also, you could never tell where you were supposed to go because the shapes were too undefined – you never quite knew where you could climb up and where you couldn't. So we designed three types of roofs 24:

- flat and completely walkable rooftops
- rooftops with obstacles protecting you from your enemies' eyes
- inaccessible rooftops

Sadly, we lost a little bit of the Asian feel and typical design but in this case proper gameplay was more important.

Readability

The third challenge was readability. Being able to instantly recognize enemies and usable objects is a must. In the beginning we struggled with this topic because we didn't have enough contrast between the different assets. For example, every wood asset mostly had the same color. There was no real eye candy drawing the player's attention. Enemies kind of disappeared within the environment because they were designed in the

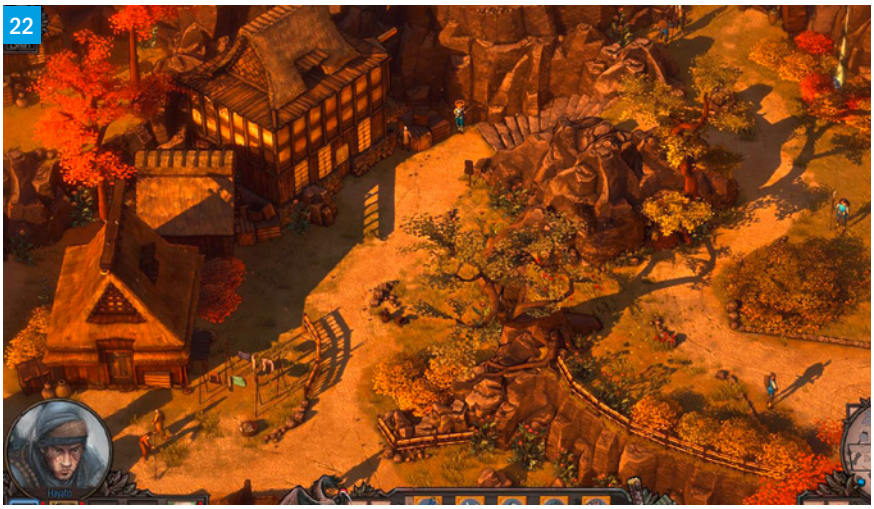
same color scheme like everything else. To solve this issue, we created some landmarks, made the shadows darker to create depth and added different color shades to give the assets more contrast 22. Then we developed a slightly different, more saturated and brighter color scheme for enemies. We also provided them with fake lighting so the environmental lighting wouldn't affect them as much as the assets surrounding them. The fake light was a gradient that made enemies look brighter from above while usually, they would look brighter on the bottom, to separate them from the ground. In this case, however, they are wearing dark trousers, so it wouldn't have had any visual effect. Thanks to the lighting, they wouldn't completely merge with the environment and were therefore a lot easier to spot.

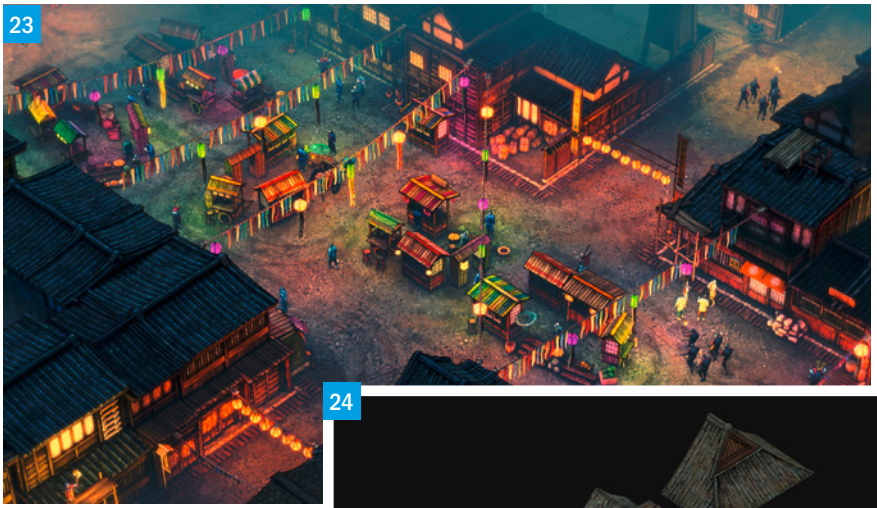
Summary

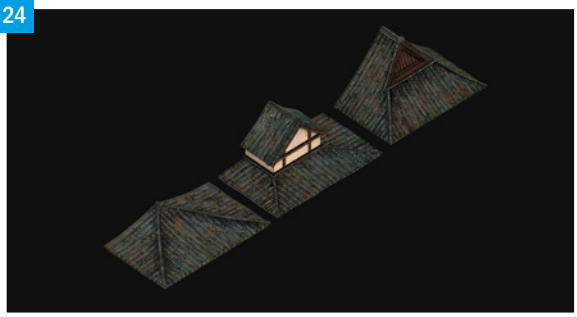
Every game needs its own process of creating a style. You will encounter different problems and will have to find individual solutions for all of them, no matter if your following project is similar to the first. Problems might occur because many reasons, a new setting maybe. I think, what's always important when creating a style, is that you use the skills your artists have. Don't try to create something you can't identify with or that doesn't fit your skills just because it would be so much cooler. When you don't create something that you can absolutely relate to, you will get stuck and be very unhappy with the result. That said, it doesn't mean you shouldn't leave your comfort zone. We definitely left ours in order to make Shadow Tactics.

After The Last Tinker and some other cute looking games we wanted to develop something different and in the early beginnings we struggled a lot – the style simply wasn't like anything we were used to. After some long discussions, reworks and iterations, however, it turned out quite well. The Last Tinker and Shadow Tactics are two completely different games with completely individual styles. Sure, we could have gone for a super realistic style, but it wouldn't have made us happy. And if that had been the case, maybe we wouldn't have put that much heart and effort into it to make it look great.

Bianca Dörr





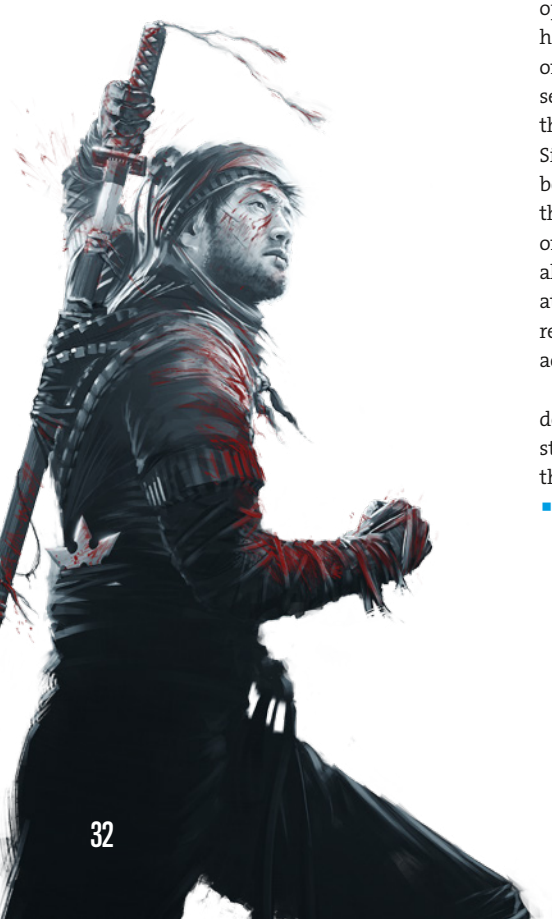


21 Time was scarce ... so was a good night's sleep.
23 24 Flat and walkable roofs, with obstacles as well as inaccessible roofs.

MAKING OF SHADOW TACTICS

LEVEL EVOLUTION IN SHADOW TACTICS

The Mimimi-way of level design: from concept to final map.



When we started working on »Shadow Tactics« we knew that the levels are one of the most important aspects of the »Realtime Tactics» genre. Especially »Commandos 2« has huge and (more or less) open environments filled with enemies that have to be conquered by the player. Most of them are memorable places in unique settings and in our opinion one of the hooks that keep players engaged with the game. Simply wanting to see that next mission can be a big factor when you are struggling with that one &%?$!§ group of enemies at the end of the current map – a map you probably already despise because you've been staring at it for hours, but also one you will fondly remember as »a pretty cool level« once you actually beat it.

So, we basically knew what we had to do with Shadow Tactics. But before we get started on actual level design stuff, let's run the numbers real quick:

- Content: the first playthrough of the game should take a player about 25 hours. After some intense mathing (the average level should take about 2 hours, so, let's divide 25 by 2 ...?) we decided that having 13 missions would be perfect. For quite some time we also worked on 3 large DLC maps. Sadly, we cut those after we had built quite a bit of them, because of...
- Time: making the game would take about 21 months, though the initial plan was 18 months.
- Team: the design team at Mimimi Productions consists of 3 people, but we had additional help from one intern for 3/4 of our production. Keep in mind though, that we also had to do other boring things like game design, creating a story and writing it, building cutscene locations, scripting the cutscenes and so on. Long story short: *Enter Trump* "Great Team. Great People." *Exit Trump*.

Well, that was that – now, let's get down to business.

The Pillars

After re-playing and analyzing old RTT classics and talking about it A LOT, we came up with three main pillars. Actually, I'm not a 100% sure if we called them pillars, but that's essentially what they were. Also, we never wrote them down and pinned them to our wall or something. But everybody knew them and worked towards them (yay, small team!). So here they are:

1. Memorable Places: each map has to be memorable and unique. If a player tells their friends about a level, they will know which one it is within seconds.
2. Huge Places: maps should be huge. Players should have to scroll over them and think

Like two different games: Below is an early design of Kage Sama's Camp, above a much more detailed version from the final game.

»Oh my ... how am I ever going to get through this?«. About 2 hours later they will have succeeded, scroll over a trail of 100 dead guards and simply feel epic.

3. Replay value: each mission should be fun when played for a second or even third time. This usually meant we had to build very openly designed maps that offered players multiple paths towards their goal. It was important to us to frequently build large parts of missions that players wouldn't visit on their first playthrough. This way they could see new stuff when coming back. Never be afraid to »lose« content when doing this – you don't! A choice only feels meaningful, if it causes the player to gain one thing in exchange for losing another.

The Concept

With these three pillars in mind, we started looking at possible settings for our levels. We wanted to deliver on the types of environments players would expect from Edo-period Japan: castles, ricefields, bamboo forests, even a monastery on the top of a mountain (ok, that one is probably more Chinese Kung-Fu movie style, but well...). We created many different levelsheets, 3 to 5 pages long documents that basically pitch a mission. Inspiration could come from anything. It might have been a larger theme like »I want to make a prison break mission!«, or a more mechanical idea like »I want to design a mission where at first you don't control any characters that can carry bodies efficiently«. The levelsheet describes the overall setting and feel and contains a simple rundown of events. It answers basic questions like difficulty, playtime and also if there are any unique features required.

A very important part of these levelsheets is a section describing each area (usually there are 4 to 5 per mission). We describe the look and feel, but also what the gameplay should ACTUALLY be like. This forces designers to think further than that cool sounding idea of »LET'S MAKE A PRISON BREAK MISSION!« and actually worry about how to implement it using the limited game mechanics they have. It also ensures that there is decent pacing to each level, since you quickly see, if you present the same type of challenge too often.

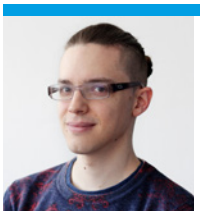
At some point we chose 13 of these levels, put them in order, made some changes that were necessary for the pacing and then started building them.

The Layout

The process for how we made our early level layouts evolved during production and today looks like this: We designers hand our ugly little drawings (or maybe even super rough in-editor blockings) and the levelsheet to Cem [Erdalan], one of our artists. We talk about the vision for the level and what is important to us from a gameplay perspective and off he goes, doing artist-things. You can quickly read about that process in Cem's section on the next page and then come back here.

You're back? Cool! So, once we have the sketch, we use it to block the level in Unity as early as possible. We don't really iterate the sketches themselves anymore, since it takes longer and it's too abstract to see all the actual problems we might face. At this stage we focus on the following aspects:

- Overall layout: how does the size of the areas feel? Do the parts of the level look interesting? Can you read it well?
- Camera: how do the camera angles work, will important parts be hard to see? The goal should be having a map that doesn't require the player to rotate the camera (which sadly never really works).



Moritz Wagner
is Design Director at Mimimi Productions.

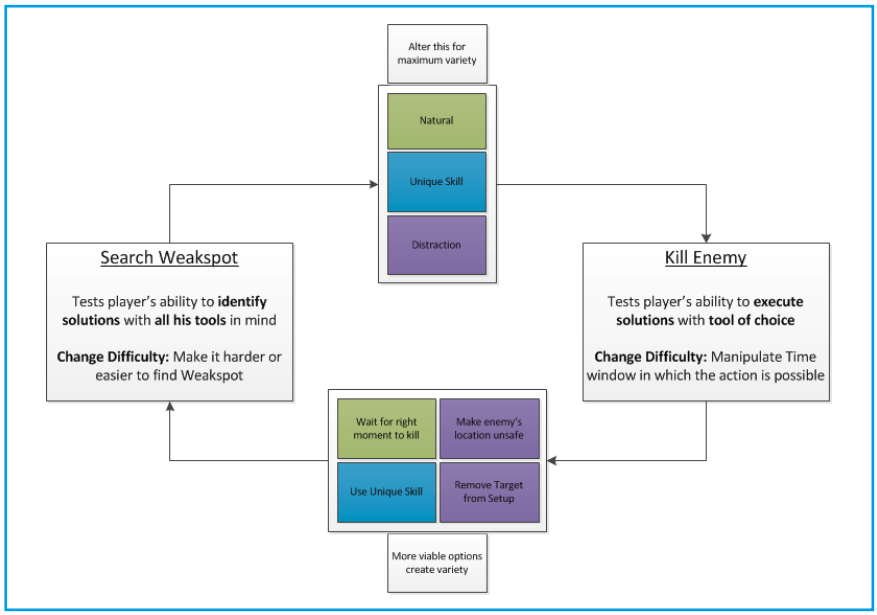
As Design Director, Moritz is leading the small but awesome design team at Mimimi Productions. His current goal is to establish »Kopfnuss« as an internationally accepted dev-term for stealth enemy setups. @corugnoll

Playing in the snow in Imai Town.

- Reality-check: is our layout plausible? Are there things that just don't make any sense in this game world?

In my opinion, this is one of the most important parts of the process. While we made The Last Tinker, we realized that using grey boxes doesn't quite do it for us. Now we use any assets that we can get our hands on.

The core-loop we defined for Shadow Tactics.



It's dirty, but very efficient. Things get scaled and abused, you see houses built out of wagon-parts and what-not. It's every artist's nightmare ... and in the case of Shadow Tactics, it came back to haunt us later. Today we aren't as reckless anymore, but hey: we were young back then and didn't care.

We talk about each iteration of these blockings with the design and art team and go over it as often as we feel is necessary. The whole process includes lots of communication between these two departments, with everyone chipping in ideas.

In Shadow Tactics I think we only had to do major layout changes to 3 of our 13 maps after this stage, which shows how valuable it is to give this process enough time and do it right in the first place.

The Gameplay Loop

Once the layout is done, the placing of enemy setups begins. During this stage we also make smaller layout changes to areas by adding cover, moving stuff around etc. The overall appeal of a level doesn't usually change, though.

RTT games are very close to puzzle games in their core loop. Players are presented with a problem and have all the time they need to analyze it. Once they have found a solution they like, they can try it and instantly reiterate and optimize their approach through quicksave/quickload (graphic, left).

The »Kopfnuss«

I talked about each level containing 4 to 5 areas before. Typically, each of these will have one main puzzle, internally we call those a »Kopfnuss«. If it's possible for a player to quickly describe this setup to a friend by mentioning 1 or 2 of its key aspects, that's a good sign.

The process of building a Kopfnuss probably is a little different for each of our designers, so I can only talk about my approach here. I think every area should have its own identity and that's what I try to find first.

I look at the environment and the vibe I want to create in that area. Should the player feel like they are in control? Should the situation feel overwhelming? Claustrophobic? Tense or relaxed? And so on. I try to find an idea that answers these questions in the right way. It could be a small scripted routine (»guards interrogating civilians«), something based on the environment (»a samurai standing on top of a tower«) or something mechanical (»4 guards looking at each other, deal with it!«). During this stage I often randomly place guards on the map and press play, just to see if something interesting occurs.

Once the identity and idea for the setup is found, I place the necessary guards, give them the routines they need and start playing. I play it over and over to find things that don't feel right, and eliminate the unwanted problems that always occur: guards looking in from outside, covers having holes for guards to see through, distances being off etc. Due to our mechanics, these setups need to be very precise and just moving or rotating a guard by a small amount can make a huge difference.

If I am happy with how everything plays, I will stop working on it and wait for feedback from the design team. After some early iteration, all team members will get to play and give feedback. Something we have started doing with our new project (which I wish we had done on Shadow Tacics as well) is that everyone records their playthroughs and uploads the video to our server. Now designers see gameplay from different player types and many different approaches. It also helps a lot when talking to someone about their playthrough. Just let the video run in 4x speed and they will remember smaller stuff and details they would forget otherwise.

How we use that feedback to build new iterations of our levels is very important. Each of our puzzles has to have multiple solutions. When we made The Last Tinker, our approach to level design was different: We watched people playing to see if they did something we didn't want them to do and then fixed that part. For Shadow Tactics, we had to change that approach and whenever we saw something unexpected, we tried to support it, not fix it. This means that at some point we »let go« of our urge to control every aspect of a puzzle. It's impossible anyway. Almost anything in Shadow Tactics can be »broken«, it's part of the game's core. Players feel incredible when they get the feeling of outsmarting designers, why should we take that from them? At some point we started thinking »Oh nice« instead of »Oh §%&$« whenever something unexpected happened and that was exactly the right mindset.

The Details

When we reached our beta milestones, we had 13 levels with good layouts and challenging

From Sketch to Concept

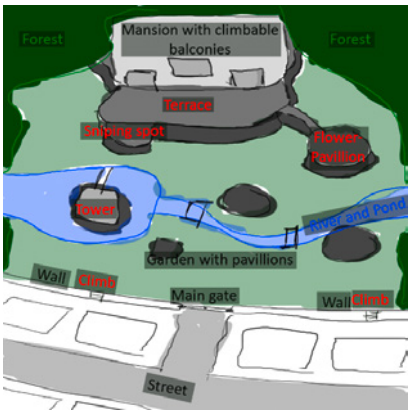
Cem Erdalan is 3D Artist at Mimimi Productions.

@cemtleman

Being tasked with the level-layout as a 3D artist certainly is rather unusual. In games like »Commandos« or »Desperados«, spending 2 to 3 three hours on a single map is absolutely normal, and Shadow Tactics isn't any different. The team knew, that because of this aspect every single screen had to offer lots of interesting elements. Therefore, creating a believable setting and level-layout was crucial, to create a rock-solid foundation to build the game upon.

In the game's early days we did a lot of research, looking for interesting historical settings in ancient Japan. We wanted the design of our 13 levels to be as diverse as possible. Next, our game designers Hambo and Mo created levelsheets including the most essential information for each map.

My job now was to create concept drawings for the blocking of the levels. I took the most crucial information right from the respective levelsheets. Prior to creating such a concept drawing, we did an enormous amount of research. We gathered tons of information about the Edo period from books and documentations. The concept for our »mansion mission« was one of my first sketches.



From ugly to pretty: Crude sketches made by designers would then be worked on further by the art department and serve as our first guidline for blocking the level.

Early beginnings vs. final game: scenes from the map »Kanazawa City« (top) and Suganuma Village (bottom).

enemy setups. At some point we also added all the story triggers etc, but I'm not sure when exactly that happened.

Now the art team came in to make everything pretty. This was also the time when the crime of reckless scaling and kit-bashing, that we had committed during the layout phase, came back to bite us. Stuff was clipping, not aligned perfectly, there were hidden walls inside of buildings because the layout had changed multiple times and so on. A lot of things had to be fixed during a moment in production when you have way better stuff to do and time is already running out. For our new project we still try to be quick during layouting, but overall we pay a little more attention to this.

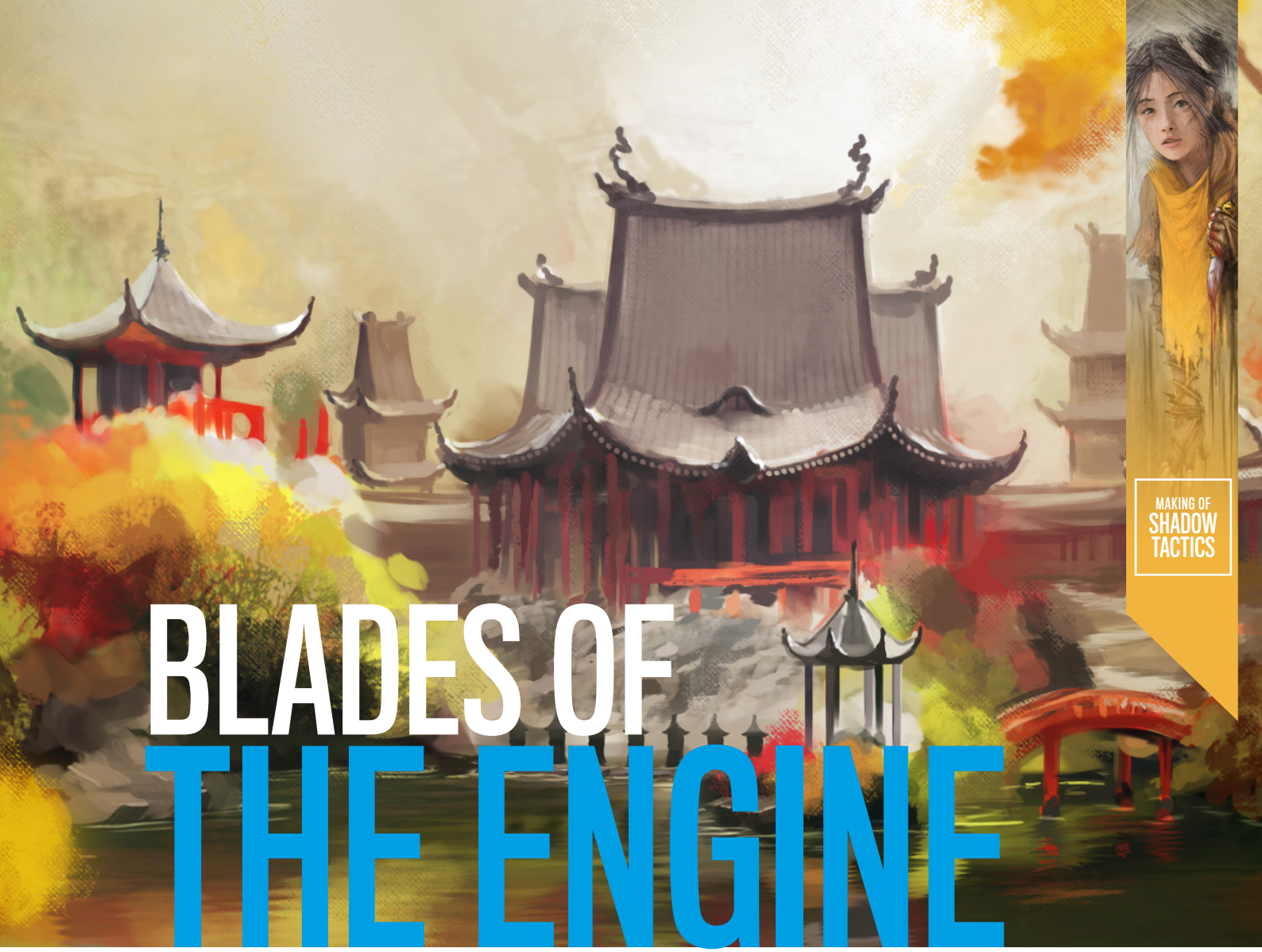
So, the art team started placing deco-assets, painting the terrain in a much more beautiful way and doing all these things that make the game look pretty. They had frequent heart-attacks when finding another one of our early layouting abominations and we had frequent heart-attacks fixing Navmesh and line-of-sight problems that occurred, whenever they moved and placed objects. But we all love each other and these small conflicts just made that love run even deeper. So much love!

We also went over all enemy setups and tried to make them feel a little more realistic. Instead of having guards standing around, being »guardy« and all that, we made them do really intense stuff like leaning against walls, talking to each other etc. This phase of polishing was very important for the final game and really helped things to come together.
Of course we also fixed a truckload of bugs during that time. There are certain assets that made baking Navmesh a total nightmare (I'm looking at you »dec_gn_plank_00«) and our jump-system turned out to be straight from hell. This daimyo dude you have to assassinate in level 5 has been broken from start to finish.
A small tip: At some level of complexity, stop trying to script stuff yourself, grab one of the coders and get that thing done properly. It will save you hours of pain, since debugging or changing scripted stuff becomes almost impossible if it's too intricate.

The End

So once we went through these stages, we just kept fixing and polishing stuff. That's all for my basic rundown of how we approached level design in Shadow Tactics. The key to accomplishing this much with such a small team was having great people and lots of honest communication. The more and earlier we talk about things, the better we can prevent problems from spreading and biting us later on. Every person on the team will have a different perspective on the levels you build. Listening to all this input and then deciding what fits and what doesn't will make your levels the best they can be.
I could write a lot more about each of the steps and go into the gritty details, but that would be a little too much. While our process was probably pretty standard, I still hope that there have been some details or small ideas you can incorporate into your own workflows. Keep building, bye bye!

Moritz Wagner



BLADES OF THE ENGINE

Tools and optimization, lessons learned and the evolution of code in Shadow Tactics.

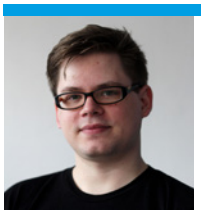
With »Shadow Tactics« now also being available on consoles, Mimimi Productions dove right into their next big project. On the following pages Technical Director Frieder Mielke and Game Design Developer Philipp Wittershagen will talk about their work on Shadow Tactics, game assets as well as important and »nice to have« features, but also about what they have learned from working on such a large-scale project and how they can transfer their newly acquired skills and experiences into their upcoming project.

Asset overview

The Asset Pipeline usually is the foundation for a lot of the in-editor work, so it has to be rock-solid. Artists should be able to drop created assets into Unity without worrying about import settings, so they can be used immediately by level designers or coders. With incorrect import settings, inconsistent naming or even just an unstructured assets folder, the natural workflow will probably be disrupted at some point. While not all of those things can be automated, you can get quite far with custom import systems based on correct naming and a sensible folder structure alone.

Unfortunately we learned this the hard way during the development of »The Last Tinker« where we had to manually change the import settings on a lot of assets based on changing technical requirements. Tracking down assets that needed the new import setting was hard and quite error-prone, and we probably missed some of them in the end.

	The Last Tinker	Shadow Tactics
Models	0.7k	0.8k (some files are whole kits)
Textures	2.2k	4.5k
Materials	2.5k	2.8k
Audio	2.2k	7k (+ 3k Japanese voices)
Prefabs	2.8k	2.7k
Scripts	1.4k	2.7k
Total	11.8k	20.5k + 3k

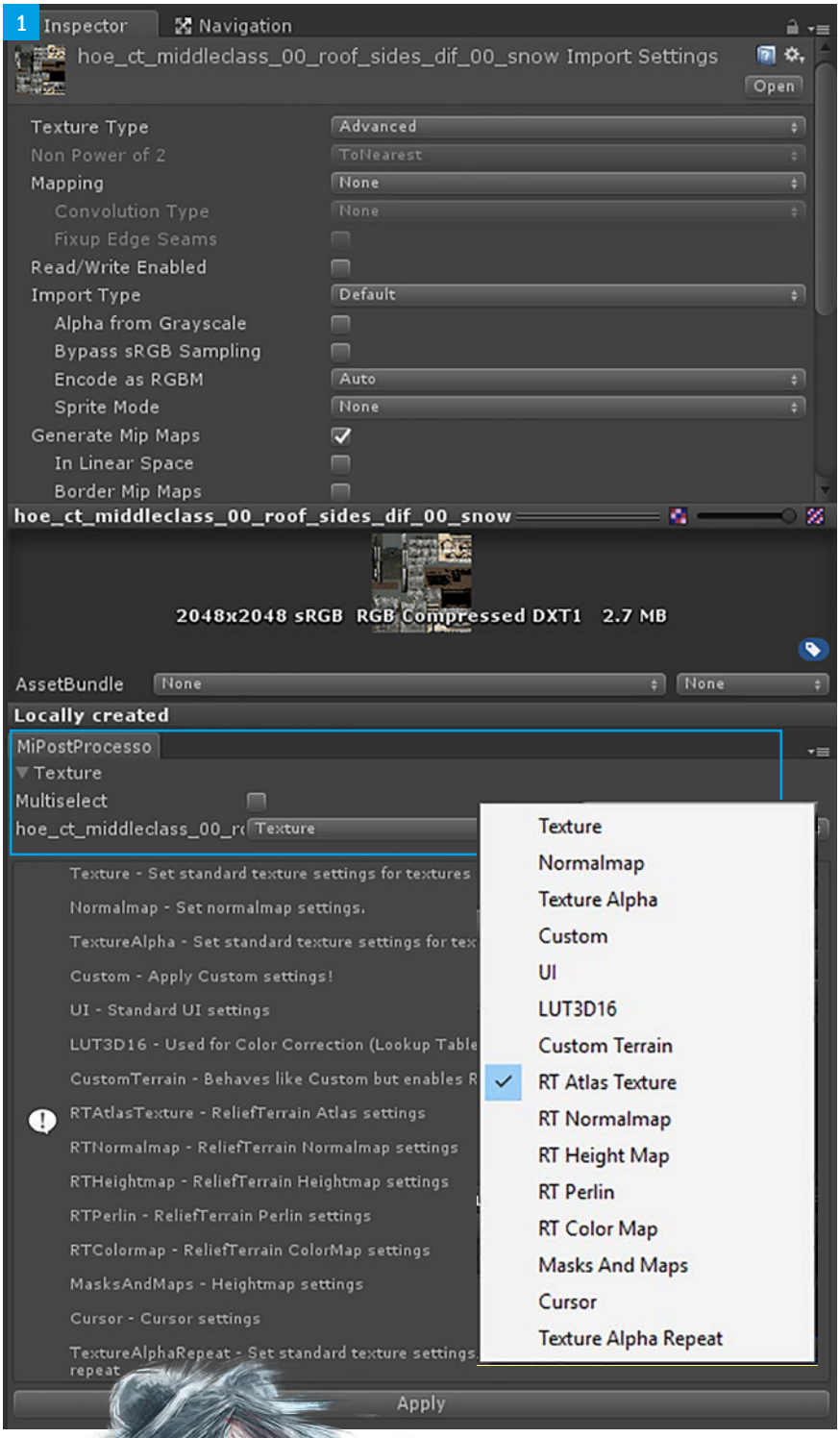


Frieder Mielke
is Technical Director at Mimimi Productions.

Since 2015, Frieder supports the team at Mimimi Productions with his unique skill-set and knowledge of AI programming and played an essential role in the development of Shadow Tactics: Blades of the Shogun. @thats_frilke

Philipp Wittershagen
is Gameplay and Tools Programmer at Mimimi Productions.

As Mimimi Productions' gameplay programmer, he supports the team with his coding skills and self-coded tools since 2015.
@phwitti



Let's take a quick look at the actual amount of assets we're talking about:

Raw Assets and Materials

As you can see, we have a lot more assets in Shadow Tactics than we had in The Last Tinker, most notably more than twice as many textures and four times the amount of audio clips. The drastic increase in audio files can be attributed to the full voice acting in Shadow Tactics, which amounts to about 5.4k files for English and Japanese each. The small increase in models stems from the fact that we used models containing whole kits for specific types of building. But in light of these numbers, when manually changing assets was bad in The Last Tinker, it was nearly impossible to do for Shadow Tactics.

Basics

The basic tools Unity gives you for importing assets are the »AssetPostProcessor« and the custom meta data you can write via »AssetImporter.userData«. If your assets follow a strict naming convention, you can do some amazing stuff with this information already. This is what we use as a base for all of our custom imports, so let me go over our texture and audio import systems in more detail.

Textures

When a texture gets imported for the first time we select a texture preset based on the name of the texture, usually via specific prefixes 1 . Unfortunately, there is no way of extending the native Unity texture type (at least none that we know of). Therefore, we set the texture type to »advanced« and handle the rest by ourselves. Although our basic import settings are similar to Unity's, this helps keeping them consistent. Additionally, we are able to change a preset easily during development and automatically reimport all textures that use this preset. There is also still the option to manually change the settings when needed by using the »custom« preset. For all other presets, manual import setting changes are overwritten immediately by our importer. The whole thing is currently its own editor window, since we have not found a way to customize the importer settings inspector so far.

Sound / AudioClips

First, we basically automated most of the suggestions found in the Unity manual. Again, based on naming conventions we know what type of audio we are handling. Primarily we distinguish between sound effects, music and voiceovers. For more advanced import settings we can also look at the length of the clips (e.g. »PCM«: This option offers higher quality at the expense of larger file size and is best for very short sound effects).

MiSfxInfo

We didn't, however, want to stop there because there is a lot more to be done with sounds than just import settings. So we developed our own audio clip container object called »MiSfxInfo« 2 . Every imported audio clip automatically creates a container for itself and based on naming we can also add variations to a MiSfxInfo. Audio clips are referenced only on MiSfxInfo assets, which are then referenced within the code, but never an audio clip itself.

In order to have everything in one place and to give our sound designers full control without having to touch scenes or gameplay prefabs, we put more and more advanced stuff into the MiSfxInfos. This includes common things like volume, pitch ranges and random settings as well as more advanced things like animation triggers, fades, max clips simultaneously played handling and emitter following.

Going further, we connected the correct audio mixer and audio source to our MiSfxInfo. In total we had 19 audio-source prefabs including 2D and 3D with different ranges, spreads, priorities etc. The importer does the following based on the filename:

- Prefix 1 sfx_gn_med_ -> audio clip import settings e.g. compression
- Prefix 2 sfx_gn_med_ -> audio mixer assignment
- Prefix 3 sfx_gn_med_ -> audio source prefab used for playing

Another benefit was that we could easily implement different spoken languages because we could switch out the audio clips referenced in the MiSfxInfo, which is the only place where the raw files are referenced.

In addition to having everything in one place, we were able to write more powerful tools for manning and editing our audio, e.g. our audio viewer which gives a good overview about which sounds are playing. It even allows easy editing of sounds settings during runtime.

Conclusions

1. **Naming conventions:** Even if your project starts a lot smaller, you can still write a pipeline later, if needed!
2. **Automatisms:** Don't to things manually, if you can have them done automatically – it saves a lot of time and is also less error-prone!
3. **Create your presets/templates:** presets and templates help you maintaining an overview. If you want to, you can still change them later!

Editor Tools: Working in Mimi-Unity

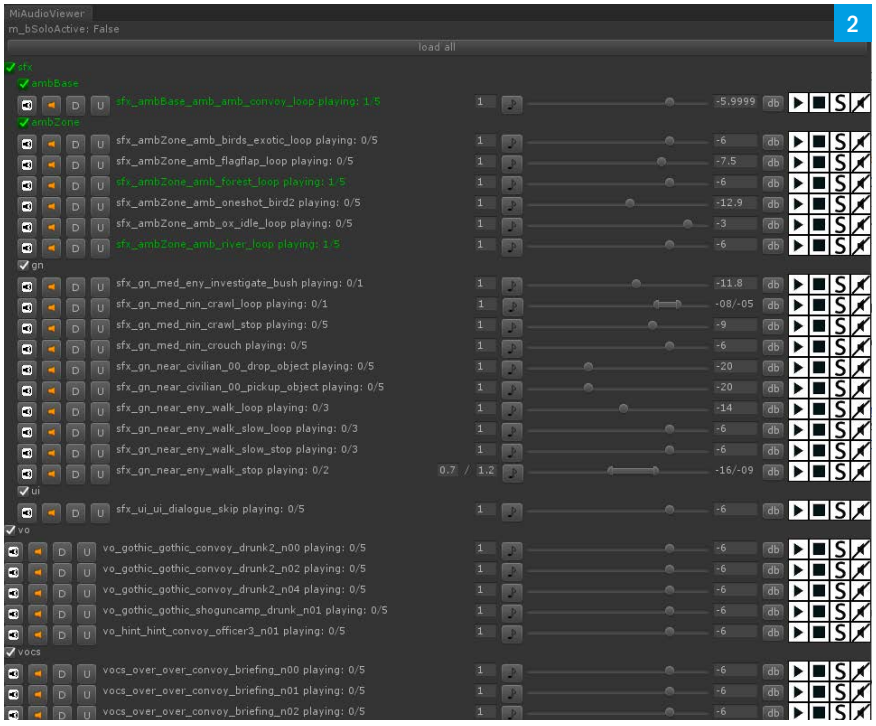
As soon as the assets are inside Unity 4, we can also start working inside the scenes. A scene in Shadow Tactics has an average of 30,000 transforms, with a maximum of 39k. There is an average of 100 NPCs per scene, with a maximum of 161 (e.g. Matsuyama City) and around 50 custom script logics. With scenes of this size, there are a few things to ensure or enforce, to not wake up in some hell of a mess of game GameObjects named anything like »CmdHighlightCharacter(Clone)(Clone)(Clone)«. Therefore we want to share some insights into our use of the Unity scene hierarchy.

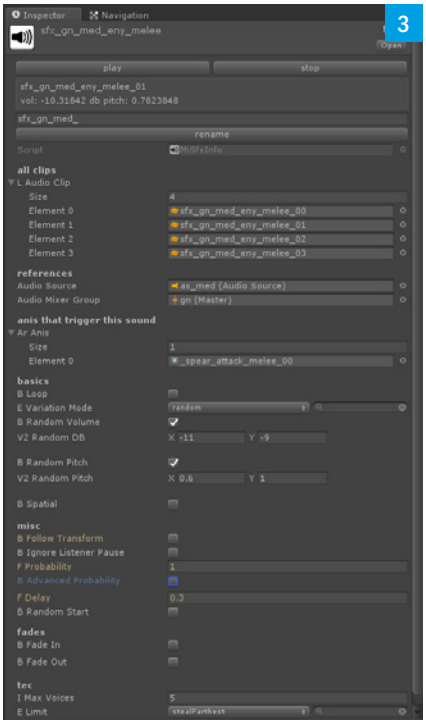
Design: How did we do it?

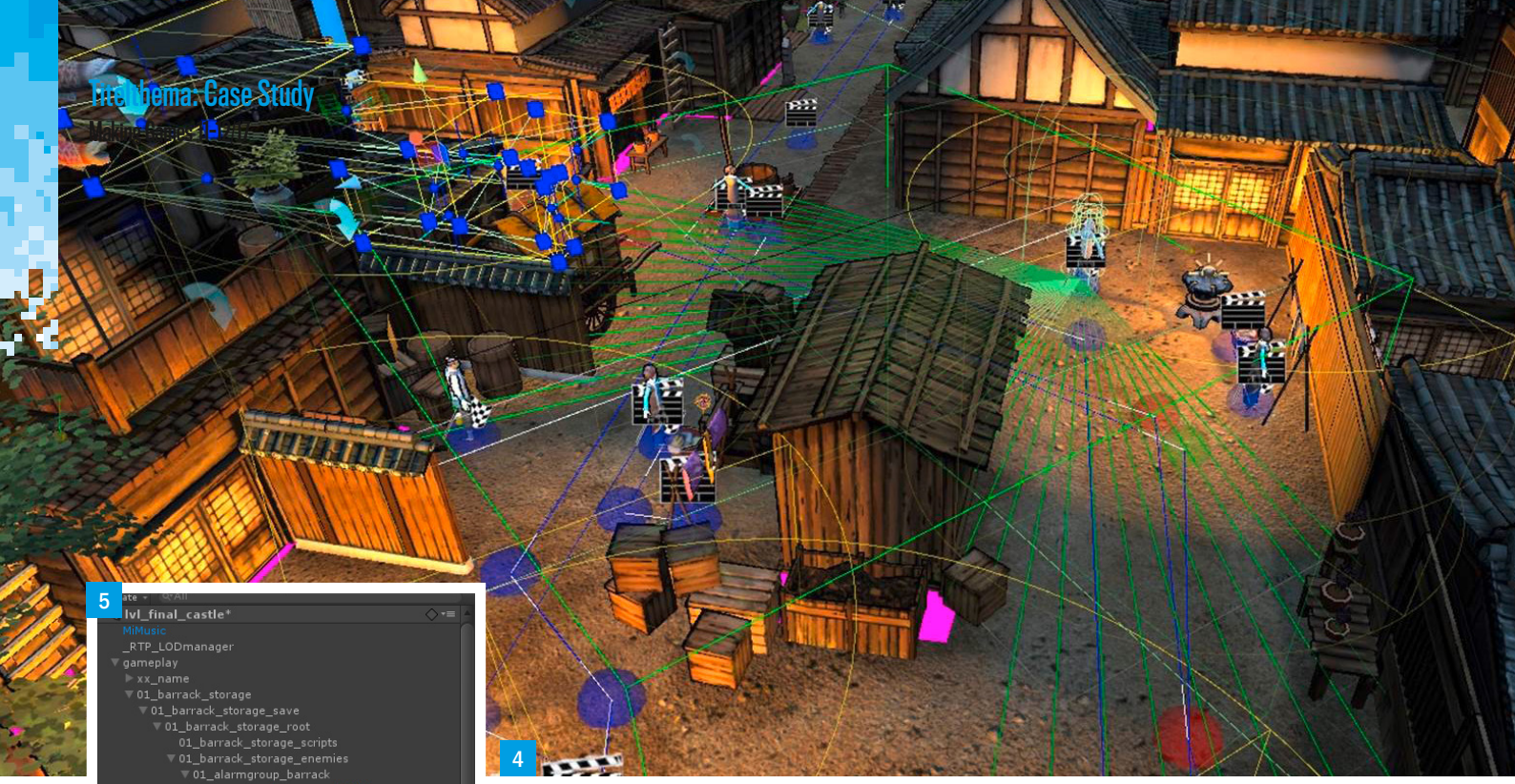
We wanted to have a very well-structured hierarchy that makes it easy to maintain a good overview over a scene by grouping as many things as possible 5, which is especially important if multiple persons are working on a scene. We had a base-structure that our game designer had to apply to, as well as some

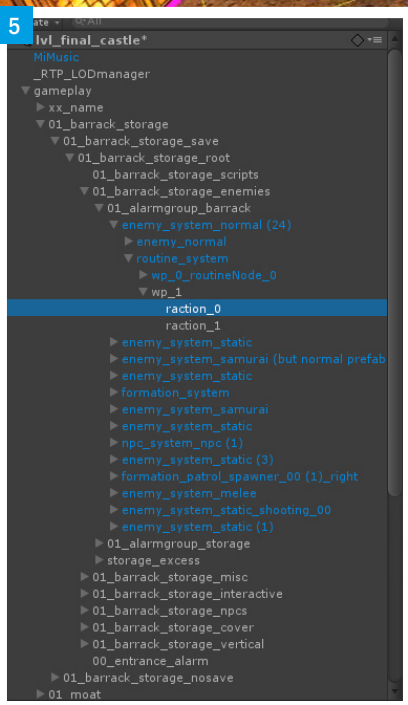
»If your assets follow a strict naming convention, you can do some amazing stuff with this information.«

2 MiAudioViewer
3 Custom audio clip container MiSfxInfo.

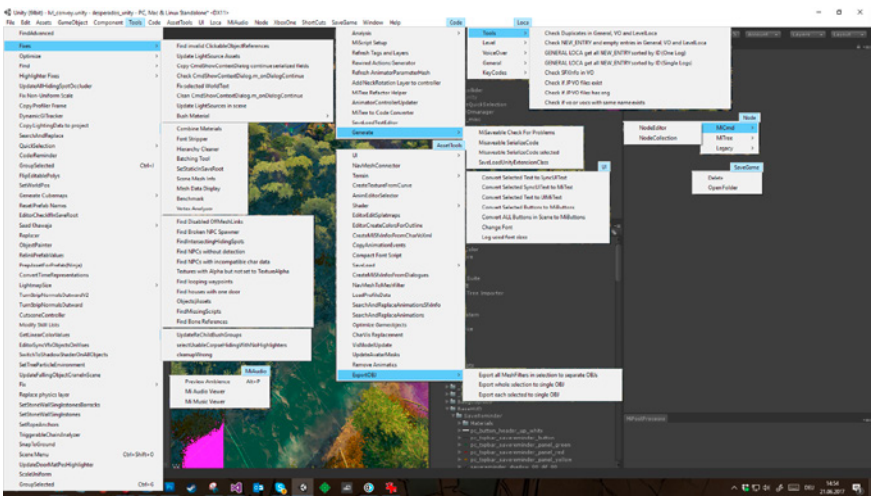








4 Imported assets in Unity: now the work on the scene can start.
5 A well-structured hierachy and grouping are the key for maintaining an overview for a scene
6 Simple, yet essential: saved transforms will win microseconds of save/load.



The editor is easily extensible, but at some point enough is enough.

»best-practices« of how things were supposed to be structured.

Main folder gameplay (0 depth)
  Level area (1)
    Save vs No Save (2)
      Enemies (3)
        Alarm enemies (4)
          Enemy (5)
            Data (6)
            Routine
              Waypoint (7)
                Action (8)

Evolution: How did it work out?

Structuring a scene in that manner and sticking to naming conventions worked very well for us – so why are we talking about this? Well, we did have a problem ... this kind of »over-structuring« using GameObjects had certain shortcomings.

The major issue here was performance: moving a transform with 6 parents is uber-slow. In the beginning we worked around it by trying to flatten the hierarchy, mostly by moving enemies up. Sometimes, however, this can still get really tricky and it isn't something you can do all the time. Another overhead is, that every »folder«-GameObject saves transform information that is not needed. Given, it's a problem that's pretty specific to Shadow Tactics, as we saved the whole hierarchy and had to recreate it when loading. Still, for us every transform we had to save, meant a few more microseconds of save/load 6.

Create Fake Folder? What we were really missing was some kind of »real« folder-structuring – some element which only exists for structuring and which doesn't really exist in

the transform-hierarchy. This is something we are currently looking into – thinking about how to fix this issue in our next project.

Use less transforms? Some of the problems actually arose from an overuse of transforms with often only one component per game object. This kind of structuring was implemented after The Last Tinker, as otherwise it's always hard to see where a reference goes as only the transforms are highlighted and have unique names. Probably we have to take one step back and see, if we can fix this issue another way, e.g. by using custom tools to help maintain a nice structure.

Editor Tools for Every Occasion

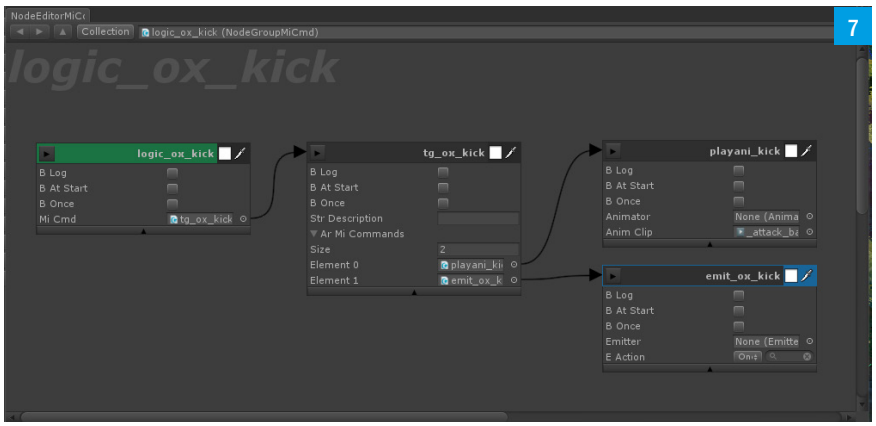
Well, 'Tools' is actually the buzzword that takes us right into the next section: editor tools. We asume that everybody knows how important tools are, no matter the project, so we won't dig into that too deep. What we want to show you though, is quickly things can escalate on a bigger project. We tried to screenshot every tool entry... however, it would have filled the whole screen, had we displayed every subentry. By now we also have a tool for finding other tools! While we prepared this article we cleaned up this whole mess by creating more categories. That was when we also realized how many one-time-use tools we had, that were simply never removed afterwards. And even after the clean-up process a lot of tools remained, in a large variety of categories.

Scene Tools

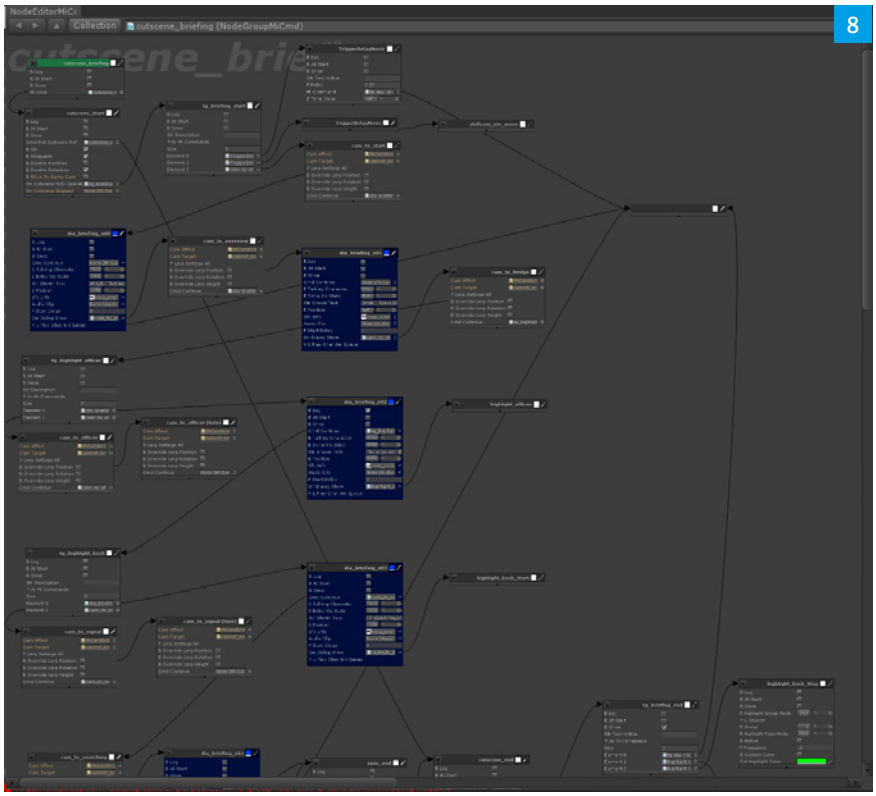
- Optimize
  - Batch meshes
  - Clean up scene
  - Analyzer
- Game design
  - Place, Replace, advanced transform operations
  - Scripting helper
- Art
  - Model editing
  - Cubemap generation
  - Particle helper
- Search
  - Advanced search
  - Super custom search scripts for fixing and refactoring
- Fixes
  - Fix stuff that level design can do wrong
  - Refactor trough all scenes...

Asset Tools

- Import/export
  - Models
  - Terrain
- Font tools
- Loca tools
- Code generation tools



7



8

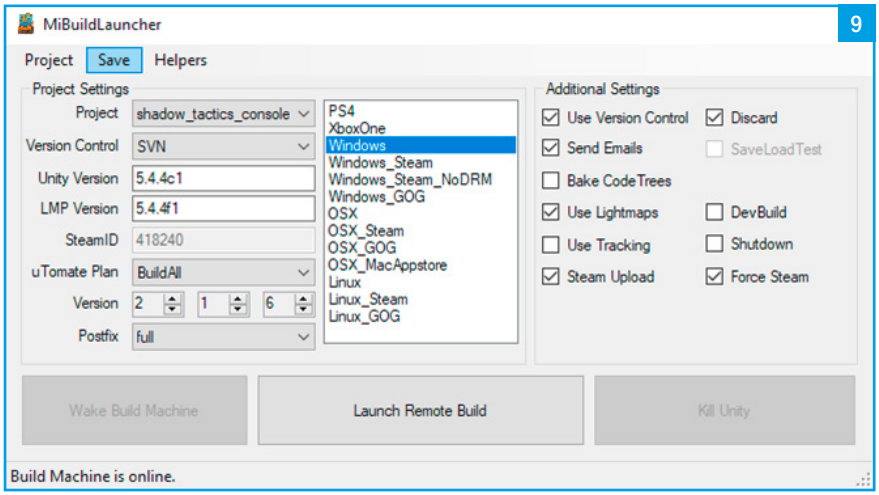
7 If the logic is triggered, it triggers two other commands in a row: one, that plays an animation on the ox and another emitting some noise.
8 A more complex example of a similar process.

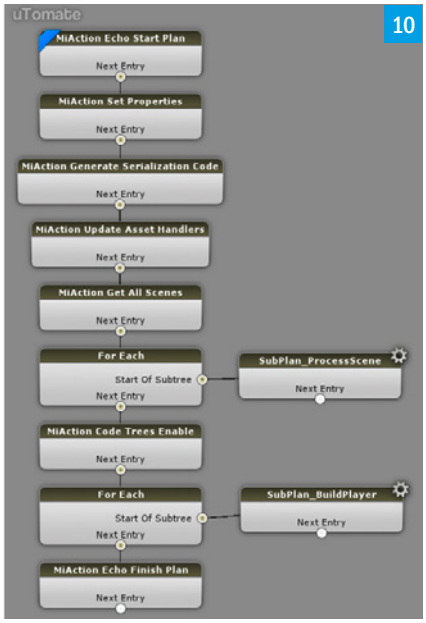
Special Tool for AI

- Routine notes for AI behaviors
  - Visualize in scene

A Node Editor for everything

In our earlier project we used our own very simple trigger system, which anonymously receives and send signals to each other. This makes it very easy to extend, however, its use is limited because it cannot forward data from one trigger to the next. Of course, to some extent our trigger system can do it anyways. The major problem was that you couldn't see from where or how a trigger got ... well, triggered and normally that info is visible. In the past, we only had a tool that would look through all the triggers and thereby for a reference to the trigger you needed the info for, so we could actually get some »chain«-information by using it. Knowing that Shadow Tactics would most likely have very complex scripting, we





9 Build launcher
10 uTomate setup

developed a node-based editor that would visualize the connection between our nodes without changing the trigger system itself. The editor is designed to display the inspectors of any script and to visualize connections, if a field inherits any from our trigger base class. As the editor is only drawing Unity inspectors, it was easy to also use it for other purposes, e.g. behavior trees, which we will talk about in the code-section later on.

The thing is: without this editor, we would have had no chance to maintain the level-logics. Unfortunately, the tool is far from perfect. It has bad performance and a lot of bugs, but so far we still don't know about another node editor that is as flexible as our model.

Conclusion: What we think everybody needs

- Create custom tools for what you do most!
- Don't underestimate the potential of tiny helper tools! A simple process might only cost you 10 seconds of your time. If you repeat this process 50 times a day, over a period of more than 2 years, it costs you around 3 days total!
- Look at the asset store: there are so many tools you can simply buy or at least get some inspiration from!
- Create a basic repertoire of tools that you can use for every project!

Creating a Custom Build Pipeline

We talked about assets and tools, now it's time to take a quick look at our build pipeline.

The need for automated builds arose early, when we needed to update certain information of a scene before we could play it. In the editor it happened automatically upon entering play mode, but we couldn't do it in a build. So we had to go through all the scenes, update their specific data holders and then start the build – an incredibly tedious task, so we automated it.

Even early on, running this simple build process could take up to an hour, so we decided to do it remotely, from a machine – our so-called »build server« – which nobody needed to actively work on all the time. From there on out we found more and more features to add to the build pipeline that would eventually include building for multiple platforms one after another, handling version numbers and even uploading finished builds to our steam repository. We do realize that there are powerful 3rd-party tools that could probably handle all of these processes, but due to the gradual development of our pipeline, there was no point in time where we felt like we needed to switch. In the early stages, there were two batch scripts that handled everything; one for sending the build information to the server, one for actually handling the build process on the server. However, after the process got too complex, we built both systems in C#.

Build Launcher

The build launcher 9 was implemented as a C# Windows Forms App, to have a simple UI that is easily extendable when new features need to be added. It has improved a lot since its first incarnation and now has lots of settings and can save settings per project. You can even boot up the build server with the press of a button! When launching a build, a remote task is scheduled on the build server.

Build Server

The build server application is implemented as a C# console application. It is the very foundation of our pipeline and it still can be started locally. This was really helpful at times, especially when windows decides to change some security settings after an automatic update and thus block the remote task scheduling commands needed to start a build. Until we figured out what was wrong, we ran the build server application manually on the build server PC.

The most important steps executed by the server application are as follows:

1. Revert and update the repository
2. Start Unity to generate code, pass #1 (CodeTrees, more on that later)
3. Start Unity to run the selected uTomate plan
4. Pack and store finished builds on network drive
5. Run Steamworks SDK to upload builds

uTomate

uTomate 10 is a tool we already used during the development of The Last Tinker, to automate in-editor workflows like bake lighting for several scenes, one after another. It is easily extendible and flexible enough to be able to handle all of our build pipeline's needs. With uTomate, we sequence all the necessary in-engine steps as follows:

The Last Tinker is a colorful third-person action adventure with an estimated playtime of 8 hours.

1. Generate code, pass #2 (Save code, more on that later). Needs compiled code from pass #1
2. Update global asset handlers (audio/input)
3. Iterate scenes
    a) Bake navmesh
    b) Run scene cleanup
        I. Delete level design helpers
        II. Delete editor data containers
        III. Update scene save info
    c) Apply lighting data if present
    d) Save the scene
4. Iterate platforms
    a) Revert & delete platform specific assets
    b) Apply custom build settings
    c) Build
    d) Post process build
        I. Copy libraries to build
        II. PS4 package creator
        III. X1 validator + symbols collector

Custom Save/Load Test

After all this is done, we usually first tested the builds by running an automated test inside the player. This was designed to spot any save/load errors in the levels, so it opened all levels chronologically and performed a few save- and load operations. By doing this before sending the build to QA, we frequently could iron out lots of the mistakes early on.

The Evolution of Code

Before Shadow Tactics, we developed The Last Tinker for PC and consoles, an 8 hour long third-person action adventure without a jump-button. Back then we had three coders:

- 1 for player and camera
- 1 for AI
- 1 additional Gameplay
- +1 »unfortunate soul« for light probes and movie plug-in

Too big to handle

During Tinker's development, our approach for dealing with all the player logic was to »divide and conquer«, as the player was too big to handle, although it was divided into a lot of components – 62 scripts in total, to be precise (camera, health, input, jump, animation, sound, effects, combat). Still, the main player script was about 5,000 lines long. The player states were implemented with enums for combat and a jump system plus some Booleans for other states:

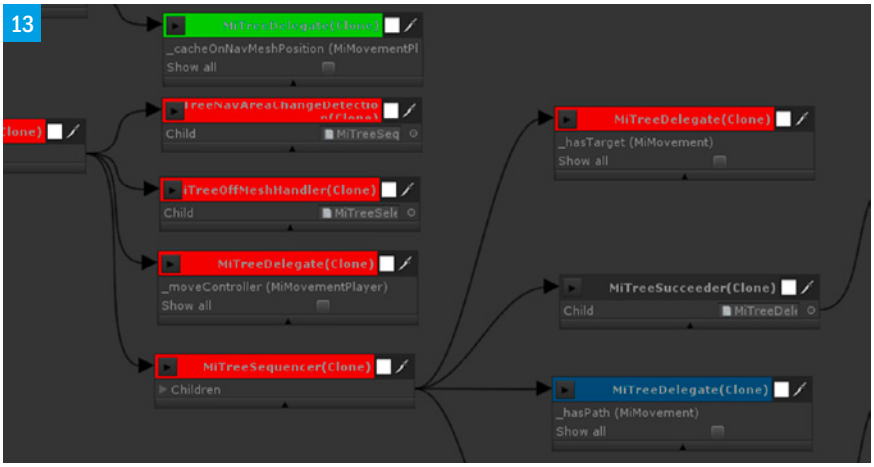
11 A more advanced AI, five different playable characters and a wide range of individual skills made the work on Shadow Tactics far more complex than on The Last Tinker.

11

```
using MiCore;

public class MiGameCreator
{
  public Genre m_genre;
  public Tags[] m_tags;

  private void Awake()
  {
    MiGame.makeGame(
      m_genre,
      m_tags
    );
  }
}
```



12 Behavior trees in MiTrees
13 Visual debugging

- The main problem here was, that we had no good overview over the player states. Bug fixing and debugging turned out to be rather complicated and the code was nearly impossible to maintain.
- For the future we were certain that we would need to find a way to force the programmer to maintain a better code structure.

Coding in Shadow Tactics

When working on Shadow Tactics, we had a different and larger team of coders, as this project was way more complex. We wanted a more advanced AI as well as different playable characters with individual skill-sets. Furthermore, player characters and NPCs share certain skills, movement and some other types of interaction. So, the risks clearly were too high to do it the same way as in The Last Tinker. Here's what we thought we'd need for more complex characters:

- Fast implementation
- Forcing a structure in the code
- Modularity (player characters & NPCs can use same code parts)
- Performance
- A visual representation

Based on these requirements we ended up implementing our own behavior tree system – our so-called »MiTrees« – which combines the benefits of visual scripting and the power and performance of classical programming. The MiTrees 12 are build using the very same node editor we already talked about in the »Tools«-chapter.

Behavior System:

We won't go into detail on how exactly we implemented these behavior trees, because they're actually not that special; we just show you all the default nodes:

- Node-States: Fail / Succeed / Running
- Composite Nodes / Control Flow Nodes: Sequencer / Selector
- Decorator Nodes: Failer / Succeeder
- Implementation Nodes / Leafs: actually only delegates to script functions
- Subtrees: to satisfy our requirement of modularity
- Initially all our AI trees were completely processed once per frame

Special Nodes: »Repeat Until Fail«

We felt our game would need some special nodes, particularly as we wanted to set up the player skills inside the behavior trees. These skills are usually build in a structure like the following:

1. Check if the player is already at the target (otherwise the movement subtree is executed)
2. Do something (i.e. remove items from the inventory/execute the skill)
3. Play an animation and wait some time
4. Do something else (i.e. stash weapons)

The key word here is »wait some time«. During some skill animations the player could not take any action. He could neither cancel the skill, nor could any input the player made – like setting a new skill and/or target – change the behavior of the character, until the skill was finished. The behavior tree was kind of meant to be trapped in a state it couldn't leave until certain actions were finished.

So, we ended up adding a »repeat until fail« node, which does exactly what it says ... and what we would later regret quite a few times during development, as the user can actually save the game, while the tree hangs in there. On load, we had to restore this exact state. We therefore had to save the complete state of all AI-trees and all of their nodes – not only some custom picked AI-states! – inside our save-games.

Visual Debugging

After using the trees for some time and being pampered with Visual Studios' debugging features (which are some of the most awesome things ever!), we realized – still early in development – that we would definitely need some kind of visual debugging. Due to all nodes

being objects, it wasn't too hard to implement, but it helped a lot. Without that, these trees were more like black-box systems that were pretty hard to debug 13.

Object Oriented (Node Type Class) Stack Iteration

As just mentioned, every tree has its own objects for the nodes inside. They are created on the basis of the tree structure set up in the project and are iterated in a stack.

Problems During Development:

- Performance
  - Execution: stack operations are pretty slow in Unity, additionally one had the default memory overhead that comes with thousands of small objects.)
  - Save/load (all node objects had to be saved – every object has overhead)
- Solution
  - Bake trees into code
    -> Custom IEnumerator.MoveNext(), a whole tree, one function
  - 6 times faster runtime! 6 times smaller save data (bath 6mb – 30mb)! 4 times faster saving (1.5 – 5.2)! 13 times faster loading (2.7 - 3.7)

Conclusion and Future Improvements

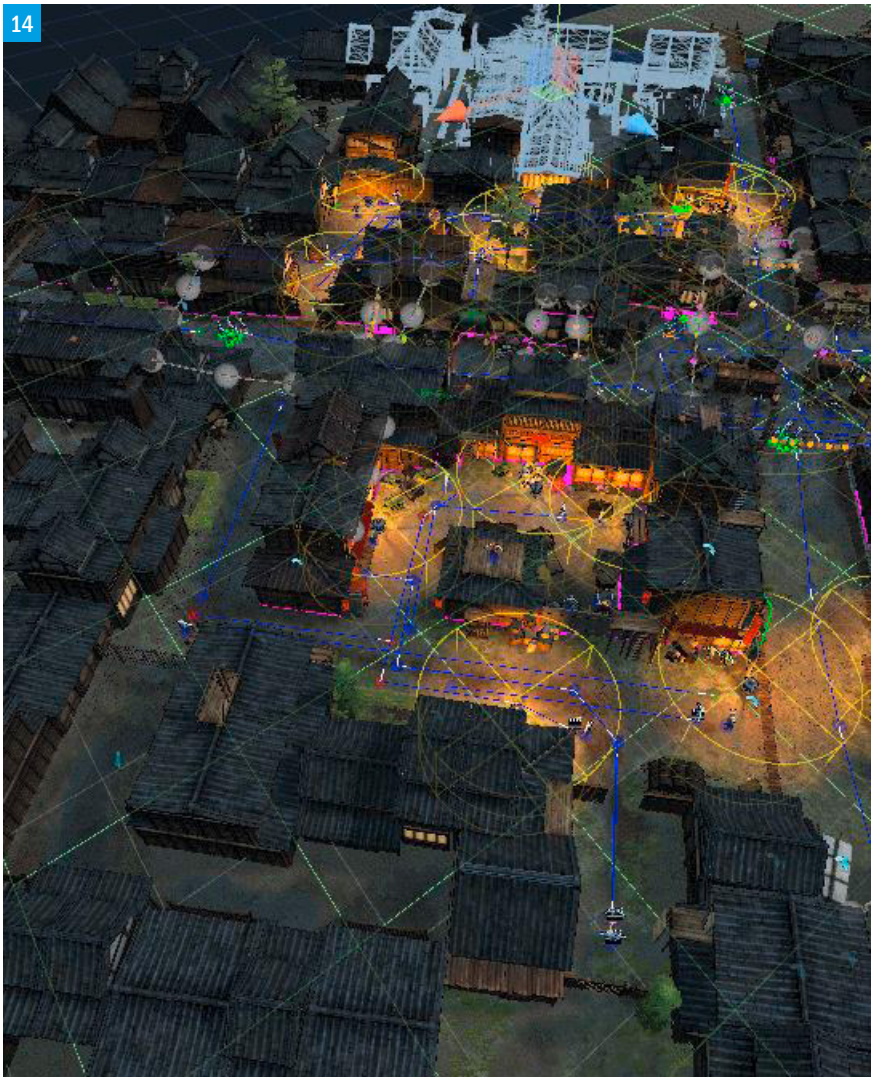
- More consistent implementation
- Better structure and division into subtrees/ blocks, now we know what everybody uses and what will change frequently
- Dynamic subtree linking -> modularity
- More back into code if nodes have dependencies
- Slows programmer down
- Make sure to not lose overview
- Improve editor!

What everybody needs ...

... is a way to maintain an overview over growing logic. We think that isn't possible with only plain code in a large project.

Optimizing Shadow Tactics for Consoles

Before we ported Shadow Tactics, we had the following situation: on our development machines [i5-2500K (4x3.3GHz), GTX 660, 16GB RAM] the PC version, that we had released in December 2016, operated at 40 to 70 fps in most cases, while in the worst case it operated just above 30 fps. The main issues we faced during development originated from AI, viewcones and physics – fields, in which we subsequently had to optimize quite a lot. In »worst case« scenarios during our first tests on consoles, the game operated with 15 fps on Xbox One and 16 fps on PlayStation 4. Quick-loading didn't really deserve its name, it easily took 30 to 50 seconds to load a savegame.

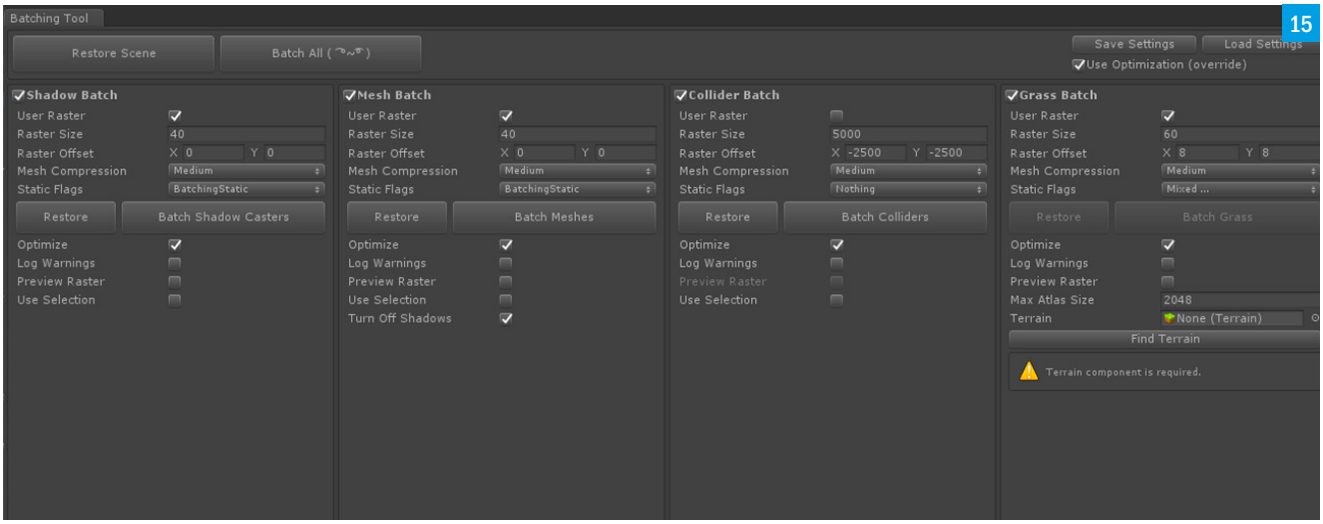


14 Custom settings forspecific batches.

So, the following points were the ones we put in most of our optimization work:

- AI performance had several passes. As we already mentioned, we baked behavior trees into the code, with huge gain in performance, but NPCs still needed too much CPU time. That's when we additionally introduced time sliced updates to the AI, which means that only 20% of all AI trees are evaluated each frame, circling through over 5 frames. As soon as an NPC leaves its idle state (detecting something out of the ordinary, performing a skill), it's updated every frame.

- Viewcones also received a few updates during development, as new features were needed or new tech was available. We removed a lot of unnecessary draw calls and render passes by setting up cameras and render layers correctly.
- In the beginning, physics were used for a lot of simple checks and we had several suboptimal collider set-ups on our characters as well. As a result we had a huge amount of constantly moving and potentially colliding



15 The number of draw calls was reduced from a maximum of 3k down to 600.
16 Hierarchy clean-up

objects. Our solution to this issue was to combine colliders, to optimize the collision matrix and to calculate all kinds of stuff in code wherever possible.

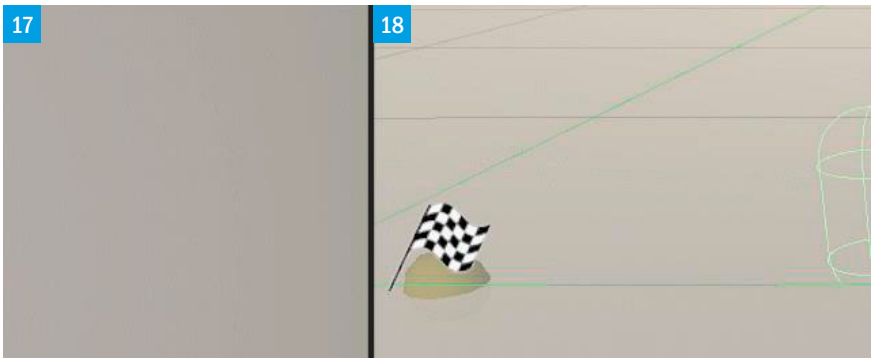
Optimization Plan

Our first tests (and those we did around various milestones) on consoles made it pretty clear that we needed to invest A LOT of thought and time into further optimizations if we wanted to keep the current visual and gameplay quality. Our goal was to go for 1080p at 30fps, while maintaining the same visual quality of a PC version with »High« quality video settings, but we knew that multi-threading core-gameplay-systems wasn't an option, since we had actually never done that and none of our systems were built accordingly – definitely something to improve for our next project!

At the same time we knew we had to look into the following matters:

- Geometry batching, to reduce CPU rendering time
- A hierarchy clean-up to reduce Unity transform overhead
- Simplified handling of off-screen NPCs to reduce logic and NavMeshAgent overhead
- Save/load optimization ... simply because we needed to put the »quick« back into »quick-load«!

17 Left: game view 18 Right: scene view



Those were the four most important points in short, now let's go into more detail:

Batch everything!

14 We managed to reduce the number of draw calls from a 2k to 3k down to 600 15. Unity has a few solid options for batching, but you can go deeper by doing everything manually. We created batches that were optimized for the specific rendering that we used in Shadow Tactics. This let us improve the various render passes and provided improved culling performance.

Now, meet the »Batcher« – a tool for creating batched geometry in scene while retaining information about original objects and meshes. With this tool it should be easy to switch between original and batched scenes to support ongoing level-changes

- »Visual geometry Batch«: shaded geometry, batched by material
- »Shadow Batch«: all shadow casters, batched by alpha test
- »Collider Batch«: since we use very few physics layers for static geometry, we create batches for each layer.
- »Grass Batch«: we baked terrain into a static mesh, so we needed to render grass manually.

After a lot of experimentation with batch sizes, mesh compression and other settings, we got most of the scenes below 1,000 draw calls and into a workable range of GPU/CPU consumption from a rendering point of view.

Hierarchy clean-up

16 Our hierarchy was pretty highly structured by using transforms as »folder« objects, therefore NPCs were deep down in the hierarchy, usually with 5 to 7 parent objects. In addition, each NPC had a quite deep hierarchy in and of itself, containing various colliders, separate GameObjects with logic scripts and of course the animation rig with all its bones.

The problem with this however is, that when a transform is moved or when you request any kind of world space information, all parent transforms have to be iterated to get the correct information on the transform you actually need. So we ...

- ... flattened the entire structure of the hierarchy (by removing parents that were not needed during runtime).
- ... unparented logic objects from the moving parts of the character. A simple guard had about 30 to 40 child GameObject that moved around with the character. Since most of the objects didn't really need to move and had a reference to their owner character anyways, it was easy to decouple them from the moving part of the character.
- ... used the »optimize game object« feature to reduce the overhead created by updating the whole rig as GameObjects and flattening it at the same time.

We still needed to keep a certain amount of structure to support the save-system functionality, but the key point was to reducing the depth and amount of moving objects that helped us free a lot of CPU resources. And by a lot, we mean A LOT – in some scenarios, these hierarchy changes gave us an increase of almost 10 fps on Xbox One!

Off-screen NPCs

This system is based on Unity's CullingGroup API, which lets you define a bunch of objects that you would like to cull depending on where a camera is looking. This would be the simplest use case, but the API is quite a bit more flexible, we just didn't need more.

NPCs in Shadow Tactics need to stay active while they are off-screen (patrol timings, corpse/noise detection, missing buddies etc.), but they don't have to be updated exactly as frequently as when they are on screen. At first, we only disabled obvious things like the animator, but down the line we added more and more systems that would operate on a simpler level when an NPC was off-screen. The biggest change, however, was the movement. We normally use Unity's NavMesh and NavMeshAgent systems to move characters. Unfortunately it gets pretty expensive when running on large maps and for up to a hundred agents simultaneously. So now, when an NPC is off-screen and idle (patroling), we move it manually every few frames along its predefined path. By doing this only every few frames, we even reduced the amount of moving objects per frame even further, which is great!

Something Unplanned: Detection 2.0

We still had problems with NPC logic costs in large scenes. The system's implementation here was quite naive: it managed a global list of currently detectable objects and every active NPC would check all those objects, if it detected them – a method that obviously scaled very poorly as the number of NPCs and visible objects across a whole scene could become quite high. We ended up adding a simple grid to each level that stored detectable objects per cell. Every NPC then only had to check the objects inside the cells in his range. This method creates a bit of overhead for populating the cells, but reduces the crazy scaling problem of the previous system by a fair bit.

After having introduced this detection grid, we were also able to use it for a few other systems that still used collider-based approaches. Reducing dynamic physics calculations is always a thing you should look out for.

Something Unplanned: Custom Update Handler

After reading into the subject of Unity's update events, we decided to try our hand at a custom update handler. We had a lot of updates that ran on various systems for each NPC, so at least getting rid of those had to be doable. We didn't want to mess with all of our systems, so the update handler is mainly used for NPC related logic now. In the end, this system allowed us to reduce around 3,000 to 4,000 update calls by only one. Checking each single thing for the need of an update was also supported by existing code, that was used for the save-system. This way we managed to cut a lot of CPU overhead, though we are not quite sure if it's Unity's safety checks or some kind of marshalling cost that makes update events this slow to begin with.

The Final Results

After having done quite a bit more than initially intended, we still have one or two areas where the frame-rate might drop below 30 fps on Xbox One. We had to reduce realtime shadow quality and a few post-processing effects, but otherwise we managed to reach our goal of 1080p and 30 fps.

Frieder Mielke, Philipp Wittershagen

FPS	Before	After
Xbox	15	32
PS4	16	36
PC	45	60
Level Load	**Before**	**After**
Xbox/PS4	2-3 min	30-60 sec (progress ani)
Save Load	**Before**	**After**
Xbox/PS4	30-50 sec	less than 3 sec

MAKING OF SHADOW TACTICS

EYE CANDY

VISUAL FX IN SHADOW TACTICS

We hardly notice them, when they're in the game, but we instantly know when they're missing and something's off – visual effects are cucial for a game's atmosphere, no matter their size.

Florian Smolka
is VFX and 3D Artist
at Mimimi Productions.

Florian has been at Mimimi Productions for almost five years. Thanks to his outstanding work on daWindci during his studies at university, he secured himself a spot within the team. Since then, his range of tasks and responsibilities grew alongside his technical expertise, which makes him the perfect bridge between art, code and visual effects. @Smou_

Shadow Tactics is a hardcore real-time stealth game in which you control a small team of specialists, each equipped with a unique skill set. Analyzing the genre, there were quite a lot of cornerstones we had to consider while doing the effects. So, let's take a look at what these cornerstones were exactly.

Visually comforting

In Shadow Tactics you generally have pretty big maps that take you from 1 to 3 hours to play through. So it's essential that these scenes feel natural and are visually appealing. The player has to feel comfortable being there, even if the situation itself is hostile.

On average, the camera is pretty far away from the characters, so you have to carefully exaggerate certain things to ensure readability and meet the player's expectations. For example: if a character uses his samurai sword to take out a target, it would look pretty boring from that distance, although it has to feel rewarding. Also, being a tactical game, the player always has to feel in charge of the situation and has to be able to quickly make out what is happening.

As we chose a realistic approach for the game, there is no black magic, just an advanced technology to suit the gameplay. One of our characters has a long range sniper rifle with insane accuracy. It's neither historically correct nor too far off, meaning it doesn't stand out in a negative way. Considering there's only limited space, I want to focus on the following aspects:

Environment:

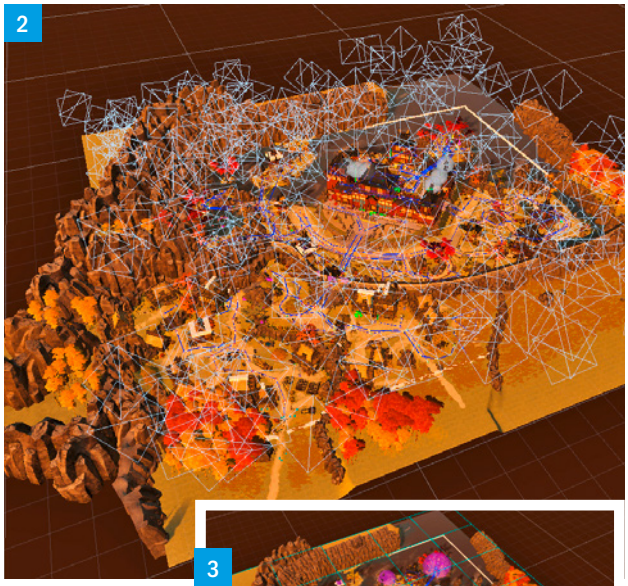
- Ambience
- Physically based

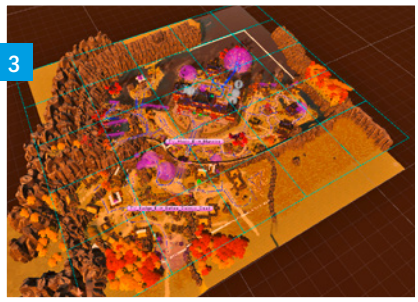
Characters:

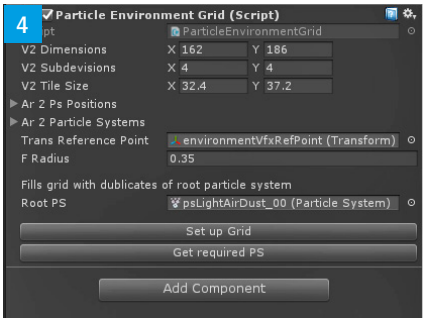
- Environment interaction
- Blood

Let's look at environment effects first, with »Ambience VFX« that are independent from the player character. 1 Constant movement on the screen is essential for creating a living scenery. You need omnipresent effects that carry the atmosphere of a level. For example, in a war zone there should be a lot of dust and smoke, maybe some ash and embers flying around. Meanwhile, up in the cold mountains, wind is blowing over the rocks, picking up and blowing away the fine powdery snow.

The thing with ambient effects like this in general is that you need a certain level of density to create an acceptable illusion. It needs to feel volumetric. A density that is too low won't either create the required effect or break the illusion altogether. So how do we achieve the density we need? Well, we could use an emitter 2 that is about the size of







4 The script behind the particle system: we set the dimensions and subdivisions, link a reference point and set the threshold to check for neighbors.

the map, but that's not really an option. Huge map equals thousands of particles at all times, but you only want the ones that you can actually see. So, what about creating an emitter around the camera? Works fine as long as you don't move it. When you move the camera, particles cannot respawn fast enough to reach the required density on your screen, especially when the cam covers great distance after rotating it. Even fast particles can't catch up. Plus, effects on the ground wouldn't work either. So we went with a grid-based solution 3. First, you create a 2D grid across the map. Then you set up a number of subdivisions. Each of the generated cells will be fitted with a particle system where the box shaped emitter matches the cell size. Then you create a reference point that is parented to the camera. At runtime, the script will ask the reference point 4 for each frame »Hey, where in the grid are you? And are there any neighbouring cells near you? If the answer is 'Yes', on turn the related particle systems and stop the ones which are currently out of range.«

It's also important to set the particle systems to »pre-warm«. That means the particle system pre-simulates a certain state. An example from Destiny 5 shows what happens if you don't pre-warm. The waterfall doesn't start to simulate until you get in range and then slowly builds up. That said, the option to pre-warm doesn't come cheap. I was skeptical at first but it turned out that the performance cost is well inside an acceptable margin.

Be Water, my Friend!

On to the next topic – let's take a closer look at a few interesting points concerning our water, which uses a second camera to create reflections.

- Pro: we get accurate reflections
- Con: it doesn't work with water on different height levels, because you would get an offset in the reflection.

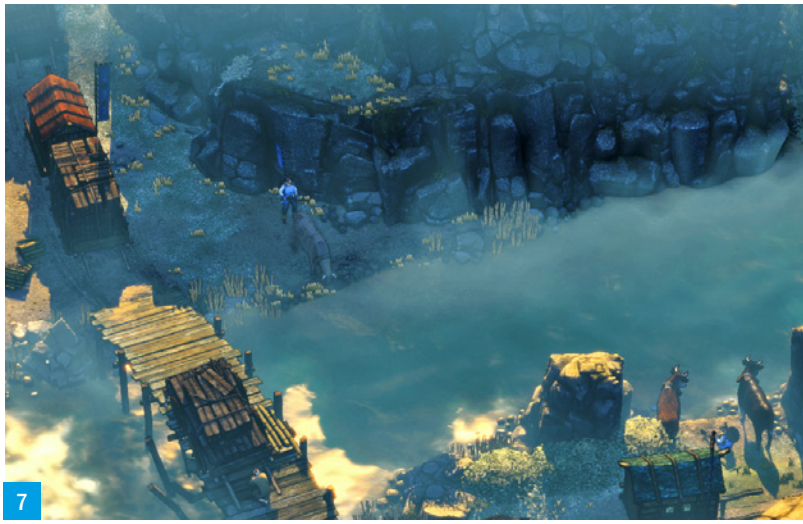
However, the large water bodies tend to be on the same level. On the smaller ones we then used a generic reflection probe. Also, we don't have a real depth fade 6; I won't go into detail here, just know it didn't work with our implementation of the enemies' view-cones. We just used a simple mask to create transparent areas around shallow spots like shores, then we put a simple ground texture underneath. Yes, we did use transparency and yes, there are shadows on the water 7. The thing is: transparency and shadows don't get along too well in Unity, which means that you cannot have shadows on transparent objects, at least without pulling some deep tech stunts.

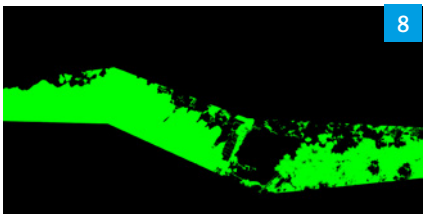
To our advantage, we already had map-sized masks in place for other stuff like blur and fog, so I just used another channel to store a top-down shadow map 8. Of course we couldn't have changed the light direction during runtime or have big dynamic objects cast shadows on the water, but that hasn't been the case, so a static map worked just fine.

Unfortunately particles and lighting don't get along all too well in Unity. However, now we had a shadow map of all the static main geometry of the level at our disposal, so we used it for some particle effects as well.

Physically Based Animations

Solid objects need to behave in a physically correct manner. Pretty much all of these animations are prebaked in Unity, using a small tool. I'll explain with the help of barrel racks 9 10. Objects are first fitted with colliders





Our water looked like this for most of the production time. Quite romantic, indeed, but there really shouldn't be any light on the water - simply because there's a giant wall blocking the sun!.

9 10 11 Physics of objects like barrels or dropping logs tend to be a bit random. Still, their behavior has to be predictable by the players.

and rigid bodies. During runtime you apply a force and let physics handle the rest. The tool will then just save each frame of the object's transformations and store them within an animation file. In this case the rack is being pushed by this long box collider with a simple forward motion to make the rack topple over.

But why go through all that trouble and prebake everything? Well, because physics tend to be a bit random. The outcome has to be predictable because every time the player interacts with such an object, it has to react in the same way 11. It's crucial that the player is in charge of the situation and able to calculate the result of his actions. For us it's important, that there are no objects where they aren't supposed to be any – obstructed pathways for example.

Destructibles – Bringing Us Joy Since ...

Destructible objects are a ton of fun, but a nightmare for developers 12 – so much work for only a few seconds of fun. Every object is pre-sliced by using Blender, but there's still a lot of manual labor involved. You have to prepare the object 13 and make sure that there are no open parts of the model. Also, I had problems with the newly created faces, the tool didn't really handle the unwrappin in an acceptable way. Eventually I ended up unwrapping every single piece by hand, because I didn't have the time to look for a better solution. Well, it wasn't fun ...

In general, every destructible object has two states 14:

- whole
- destroyed, fragmented

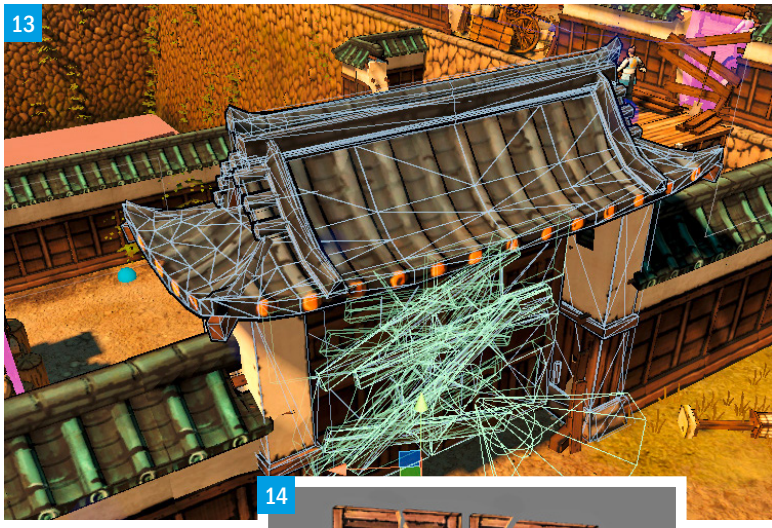
When an object is being destroyed, you simply switch the states and play the prebaked animation.

Of course, later on you don't have the colliders anymore, that's why we prebake after all. The gate 13, for example, has a »pusher« object – just like the barrel rack – that drives the other objects in a certain direction. Explosion-like setups are relatively easy to make because every rigid body can be turned on from the beginning. With the force being applied instantly, the objects don't really have a chance to just drop down like they normally would. Cascading motions, however, are much harder to achieve. If all rigid bodies were turned on and gravity was affecting them from the beginning (which they have to), they would simply fall down. So, I wrote myself a script that is applied to each rigid body and checks its neighbours for movement. If any of them move over a given threshold, they switch themselves on. Now, when you push away your first object, it cascades from that point over the whole object.

Character Related Effects

Next topic: visual effects that are related to characters within the game. First, we will take a look at environmental effects that are triggered by players. Characters always need to feel like they're connected to world. They should blend in – of course not to the extent that you cannot see them anymore, but just enough to make them look like they belong to where they are. Adding effects that simulate interaction with the


12


13


14

Destructibles are a joy to look at, but a real pain to create.

environment helps binding them to the world, but it brings us to another problem. When your character stirs up dust while walking, for example, you'd also expect splashes while running through water; the same goes for snowy areas 15. If that doesn't happen, it can easily reflect badly on the immersion. That's where terrain-dependent VFX come into play. Unfortunately, that's the kind of stuff people hardly notice when it's there – but it would look and feel strange, if they weren't. Nobody will come up to you and say »Hey dude, what an amazing job you did on these dust clouds. They're absolutely sensational!« Still, most of the times it's worth going the extra mile for stuff like this, but once you start there's literally no end 16. You stumble from one tiny detail into the next one. Fun fact: While preparing this article one of my colleagues came up to me and actually said »Wow, I didn't know that there were different effects for that.« So much for people noticing.

Let There Be Blood!

Talking about complex issues, let's move on to »blood«. Blood is a fluid and fluids tend to behave in a complex way. When approaching a (fluid) substance like blood, it helps to break it down into its fundamental components and think about their defining characteristics.

Let's start with a simple blood splash particle 17. First, I set up a fluid simulation in Blender and created a nice splash shape. Later on, however, we didn't just want this particle to fade out to make it disappear. It was supposed to dissolve in a more natural way, like getting thinner and reshape into small drops 18. So, in order to create kind of a dissolving motion, we needed to make a base for a treshold animation in Unity.

For those who don't know how it works, please let me explain real quick: At the beginning, every pixel that carries a tiny bit of color information will be shown. Over time you raise a value from 0 to 1 and cut away everything that's lower than this value at this very moment. So, the »brighter« a pixel is, the longer the texture will be shown at the same position (To see how it works in detail, please check out this website: http://www.alkemi-games.com/a-game-of-tricks-iii-particles-fun-part1/)

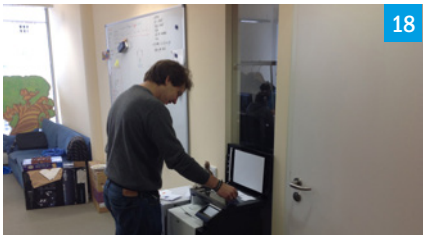

15


16


16


17


18


18


20

19 20 Drawing blood splashes by hand and making them look realistic would have been very difficult. So, Florian reused some assets he originally created for The Last Tinker, back in his early days at Mimimi Productions. In order to create some special effects, he let watercolors drop on paper, then he scanned the stains. These assets never made it into The Last Tinker, but worked perfectly for Shadow Tactics. The lesson learned: never throw away stuff you created, just because you didn't have any use for it yet - it might come in handy some day!

So, I created a mask by spawning metaball particles inside the shape in Blender and animating their size over time with a bit of an offset. What gave me the idea was a talk by Eben Cook about the Blood in »The Last of Us« at GDC. Of course he had different tools like Houdini at his disposal, but it gave me a helping push into the right direction.

If we now take what we already have, throw it into Unity and write a small shader, we're off to a good start, but it's obviously not enough. Liquid that sprays into the air splits into small droplets that shape and shift around and – due to constantly changing surface angles – reflect the light in absolutely crazy ways. To simulate this effect I could have used a normal map but that would have eventually been sort of an overkill for such a small splash. Plus, it would have taken up valuable memory space. So I used a simple dotted texture that scrolls around in world space really fast. It's actually a bit stronger around the edges for which I used another mask, but we can ignore that for explanatory purposes. The good thing is that you can't really make out what's happening in detail while playing, because everything happens really fast, but it does a good job simulating the sputtering of droplets in the air.

We need ... more blood!

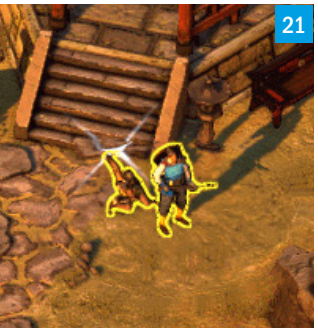
Yuki's melee 21 attack was actually the first I created because it has a nice flow to it and was therefore perfect to test things. Once I was finished with the first draft I showed it to my colleagues and they said »Ok, this looks nice, but where's the blood? There needs to be much more of it!« And I was like: What did they just say? More blood? Well, you don't hear that every day – but they were right. I first went with a normal amount of blood which was hardly visible from further away, so I had to find the sweet spot between »unrealistic« and »convincing«.

To better illustrate the impact of an attack we used some blood »decals« on the floor. The reason why 'decals' is put in quotation marks, is because we didn't have actual decals. They were just simple quads spawned just a bit above the ground. Whenever one of these blood splashes spawns, a script roughly checks the quad's bounds for ground underneath – if there's none, then the splash will not be shown. If the check results come in positive, it will pass the normal version of the surface underneath to the quad and rotate it accordingly. Spawn and despawn are basically done with the same technique that I mentioned before, only with a softer threshold and different maps.

No Gore Fest

We never wanted our game to be a virtual bloodbath. The use of blood was always meant to be justified and to give the game a realistic touch, at least to a certain degree. It was important for us never to use blood for the sake of brutality, there had to be a reason for its use. Like the scene in which a samurai who has brought great shame over his daimyo, is asking for permission to commit seppuku (ritual suicide) and thereby restore his honor – that's pretty much as bloody as it gets in Shadow Tactics. Yes, it is brutal and messy BUT slicing open your stomach and be being decapitated afterwards ... well, that's exactly what it is: messy and gruesome. It simply underlines how desperate someone must have been to go through with an act like this. So much from me about VFX in Shadow tactics – hopefully I did not bring shame over my house.

Florian Smolka


21

Yuki's melee attack from left to right, from afar and from a close distance.

THE SHOGUN'S BLADES

WATASHI-TACHI WA NINJA DESU

Mit fünf Spezialisten gegen den Shogun und seine Armee - vom Werdegang der Assassinen, ihren einzigartigen Fähigkeiten, Persönlichkeiten und Geschichten.

Wer sich einst durch Echtzeit-Strategiespiele wie »Commandos« geschlichen und geplant hat, der weiß genau um die Bedeutung jedes einzelnen Teammitglieds. Wollte man die bockschweren Missionen meistern, war man voll und ganz auf seine Spezialisten angewiesen, und jemanden im Kugelhagel zu verlieren war gleichbedeutend einem »Mission gescheitert«. Bereits nach wenigen Spielstunden gingen einem die Fähigkeiten der Commandos in Fleisch und Blut über. Den Green Beret schickte man bald instinktiv vor, doch genauso wusste man, wann man den Pionier lieber erst nach Landminen suchen ließ. Darüberhinaus erkannte man jeden der fünf Charaktere bereits an seiner Art »Yes, Sir!« zu sagen.

Als es darum ging, ein neues Ensemble für Shadow Tactics zu entwerfen, war uns klar, dass auch wir dies voraussetzen mussten. Unser Fokus lag darauf das Gefühl zu erzeugen, echte Menschen mit starken, individuellen Persönlichkeiten zu spielen. Jeder Charakter musste einzigartig sein und seinen ganz speziellen Teil zum Gelingen der Missionen beitragen. Da wir noch nie in diesem Genre gearbeitet hatten, gab es natürlich jede Menge Fragen: Welche Archetypen gibt es? Welche Skills muss jeder Charakter beherrschen? Welche Persönlichkeit sollen sie haben? Wie bringen wir Story und Gameplay zusammen? Und vor allem: Wieviele Charaktere brauchen wir überhaupt?

Eine Handvoll Ninjas

Zu Beginn stand die Entscheidung, dass wir fünf Charaktere brauchen – nicht mehr und nicht weniger. Im Vorfeld nahmen wir alte Titel wie »Commandos« und »Desperados« unter die Lupe genommen, um ein Gefühl für das Genre zu bekommen. Bei unserer Analyse kristallierten sich schnell diverse Charakter-Archetypen heraus. Auch bei der Verteilung der Skills galt es verschiedene Details zu beachten, beispielsweise wenn sich Skills überschnitten oder auch wenn Basisfähigkeiten wie der Nahkampf oder das Tragen von Leichen nur auf einen Charakter gelegt wurden.

Alle der genannten Titel hatten jedoch mehr als fünf Charaktere, darum waren wir uns trotz der Recherche ein wenig unsicher bei unserer Entscheidung. Im Nachhinein war es die richtige, wie uns auch der ehemalige Desperados-Entwickler Ralf Adam bestätigte. Von da an war unser Ziel, ein möglichst vielfältiges Team zu erschaffen. Jedes Mitglied sollte eine bestimmte Stärke besitzen und diese auf Anhieb ersichtlich sein. Spieler sollten sich nie fragen »Was mach ich denn mit dem?« oder »Wieso ist der überhaupt in der Mission dabei?« Bei der Entwicklung stand das Gameplay für uns an erster Stelle. Wenn sich das Team nicht gut spielen ließ, dann würde es die Story am Ende auch nicht

Martin "Hambo" Hamberger
ist Game und Level Designer bei Mimimi Productions.

Martin ist seit der Studiogründung Game und Level Designer bei Mimimi Productions. Darüber hinaus hauchte er als Storywriter sämtlichen Charakteren aller bisher erschienen Studiotitel Leben ein.
/martin.hamberger.9

Assassinen damals: Neben Mugen und Aiko bestand das ursprüngliche Ensemble aus einem Ninja-Gebrüderpaar sowie dem im Rollstuhl sitzenden Alchemisten, der mit Bomben und Gasgranaten hantierte

Assassinen heute: Anstelle des zweiten Ninja gesellte sich Straßendiebin Yuki hinzu, während der frühere Alchemist Takuma nun auch mit Gewehr, Marderhund und Holzbein eine gute Figur macht.

mehr rausreißen. Daher war uns auch klar: Die Story kommt zum Schluss! Natürlich sollten die Charaktere über den Spielverlauf eine Persönlichkeit entwickeln. Auch wollten wir eine Hintergrundgeschichte, die aus dem Gameplay heraus entstand. Da gab es allerdings ein paar Dinge zu klären – beispielsweise wie man zu Personen steht, die im Laufe des Spiels zwangsläufig mehrere dutzend Wachen abstechen werden ...

Töten gehört nun einmal dazu

Es ist schon ein seltsames Gefühl, wenn man nach kindgerechten Games wie »The Last Tinker« oder »Oh, wie schön ist Panama« zu einem Projekt übergeht, in dem Schleichen, Meucheln und das Verstecken von Leiche plötzlich auf der Tagesordnung stehen. Da gab es Diskussionen wie man mit der Gewalt umgehen sollte und wie unsere Charaktere oder – noch viel wichtiger – der Spieler dazu stehen würde. Wir wollten nicht in die Rolle eiskalter Mörder schlüpfen. Wir wollten Charaktere verkörpern, die für ihre eigenen Ziele und Ideale kämpfen. Am Ende beschlossen wir, die Grenzen klar abzustecken: Es nicht unbedingt schön, aber das Töten gehört leider zum Job. Unsere Protagonisten wissen und erkennen das an, jeder auf seine Art. Die Gegner sind stets in der Überzahl, und die Leute gegen die man als Spieler vorgeht, haben durch die Bank Dreck am Stecken. All das hat uns geholfen den richtigen Ton zu finden.

Das Setting unserer Wahl – Japan in der Edo-Periode – kam uns natürlich sehr entgegen: Samurai, Ehre, Verrat und blutige Intrigen – das war unser Milieu. Wir konzentrierten uns auf starke Typen wie den geschickten Ninja, die tödliche Geisha und den alten Greis, der mehr ist, als er auf den ersten Blick zu sein scheint ... alles Charaktere, die schnell greifbar waren und denen wir im Verlauf des Geschichte zusätzliche Tiefe geben konnten.

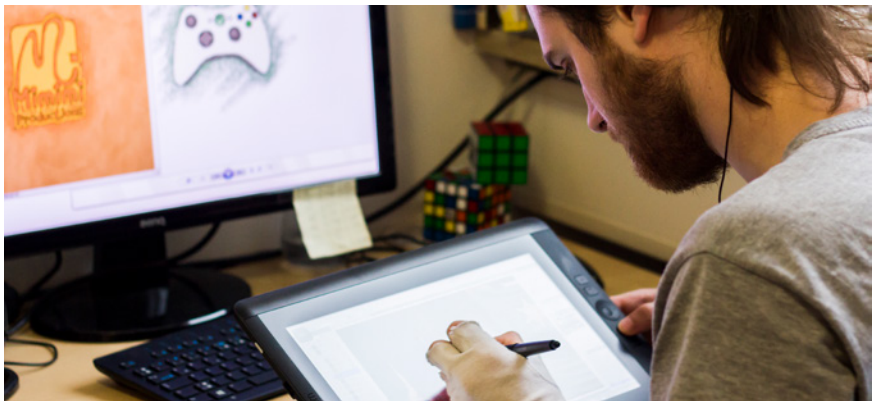
Wir wollen keinen Fahrer!

Als wir anfingen die Skills der Charaktere zu designen, hatten wir ein klares Ziel vor Augen: Die einzelnen Mitglieder des Teams mussten gut miteinander funktionieren, jeder in der Gruppe sollte über den gesamten Missionsverlauf nützlich sein. Es ertönte immer wieder der Spruch »Wir wollen keinen Fahrer!« Der Fahrer war ein Charakter, der im alten Commandos immer nur blöde herumstand, bis er am Ende der Mission den Fluchtwagen lenken musste. Das mag etwas übertrieben klingen, erklärt aber ganz gut, worauf es uns ankam.

Anfangs wollten wir das Zusammenspiel zwischen einzelnen Teammitgliedern zu einem zusätzlichen Feature machen. Ähnlich wie im späteren Desperados sollte es Aktionen geben, die nur von mehreren Charakteren ausgeführt werden konnten, beispielsweise eine Räuberleiter bilden oder ein unscheinbares Liebespaar mimen. Diese Ideen waren sehr speziell und wurden schnell wieder verworfen. Am Ende haben wir die Skills offen designed und den gleichzeitigen Einsatz mehrerer Figuren durch spezielle Timings im Gegnerverhalten und Features wie den »Shadow Mode« unterstützt.

»Die einzelnen Mitglieder des Teams mussten gut miteinander funktionieren, jeder in der Gruppe sollte über den gesamten Missionsverlauf nützlich sein.«

Bis es allerdings so weit war, gingen sämtliche Skill-Sets durch einen Iterationsprozess.



Hinter den Entwürfen von Concept Artist Lucas stehen mitunter prominente Figuren wie The End aus oder Schauspieler Ken Watanabe.

Neue Fähigkeiten wurden implementiert, getestet und angepasst. Dann wurden die ersten Levels gebaut und wir konnten Erfahrungen mit Gegnerverhalten sowie dem Zusammenspiel der Charaktere sammeln. Viele Ideen mussten wir verwerfen, weil sie entweder nicht funktionierten oder einfach nicht ins Spiel passten. Nichts Ungewöhnliches, gerade wenn man sich an ein neues Genre wagt.

Weniger ist oft mehr

Beim Designen der Skills gab es noch einen zusätzlichen Faktor zu beachten: Jede Fähigkeit und jedes Feature musste sowohl mit einem Controller als auch mit Maus und Tastatur steuerbar sein. Das schränkte uns anfangs natürlich ein, kam dem Design letztlich aber enorm zugute. Auf diese Weise waren wir dazu gezwungen, uns auf wenige distinktive Skills zu fokussieren, die sich unkompliziert bedienen ließen und trotzdem ausreichende Freiheit für kreative Problemlösungen garantierten.

Anhand dieser Vorgaben entstand schließlich unser Template für Skill-Sets. Jeder Charakter erhielt zwei Signature-Skills, die ihn definieren und die man immer im Hinterkopf hat, wann immer man sich mit einem Gegner-Setup konfrontiert sieht. Wir wollten erreichen, dass Spieler in solchen Situationen sofort denken »Klar, hier brauche ich den Samurai, da er die gesamte Gruppe mit einem Hieb abräumen kann, während der Ninja sie ablenkt.« Jeder unserer fünf Charaktere folgt grob diesem Signature-Template und erhält je einen Special-Skill wie den Shuriken, den Multikill, die Falle oder die Verkleidung sowie eine weitere Fähigkeit, um Gegner abzulenken; dazu gehören der Steinwurf, die Flöte oder eine Sakeflasche. Dazu gesellen sich je ein Nahkampf-Kill, eine versteckte Schusswaffe und eine Heil-Fähigkeit für jeden Assassinen. Diese Skills unterscheiden sich von Charakter zu Charakter allerdings nur sehr gering, da sie quasi zur Grundausstattung gehören. Für deutlich mehr Abwechslung sorgen die passiven Skills. Agile Charaktere wie Ninja Hayato oder Diebin Yuki können klettern und auf Hausdächer springen. Der starke Samurai Mugen kann dafür schwere Objekte oder gar zwei Leichen auf einmal tragen und die stärksten Gegner im Spiel im Alleingang töten.

Natürlich experimentierten wir auch mit aktiven und passiven Skills: Greis Takuma, der Scharfschütze des Teams, ist weder stark noch agil, er besitzt auch keinen Nahkampf-Kill. Dafür sind sein Gewehr, seine Granaten und sein Tanuki, der kleine japanische Marderhund, der auf Kommando Gegner ablenkt, im Vergleich zu allen anderen Skills sehr mächtig – sofern man weiß, wann und wie man sie einsetzt.

Zwei halbe Ninjas

Wie bereits erwähnt, haben wir beim Iterieren viele unserer Skill-Ideen verworfen, was sich wiederum auch auf unser ursprüngliches Charakter-Lineup ausgewirkt. So hatten wir zu Beginn beispielsweise zwei Ninjas im Team, die sich im Verlauf der Handlung als Brüder herausstellen sollten. Die Grundidee war, dass einer der beiden nur töten, der andere nur klettern und infiltrieren konnte. Wir mussten uns aber recht schnell eingestehen, dass diese Entscheidung hinsichtlich des Gamesplays Unsinn gewesen wäre und aus dem Art-Department auch regelmäßig mit dem Spruch »Was ist denn der für ein Ninja, wenn er nicht klettern kann?!« quittiert wurde. So wurden aus unseren zwei halben Ninjas ein ganzer und wir hatten wieder einen Slot frei. Den belegte schließlich unsere junge Straßendiebin Yuki, der wir das Fallensteller-Skill-Set verabreichten, das wir eh noch irgendwie unterbringen wollten.

Der alte Sniper war da schon etwas weiter, allerdings kam der in den ersten Concepts noch im feudalen Rollstuhl daher. Seine ursprüngliche Bezeichnung war »Alchemist«, denn er sollte sich Gasgranaten, Gifte und Bomben zu Nutze machen. Mit etwas mehr Erfahrung wurde uns allerdings klar, dass der Rollstuhl zu problematisch für Gameplay und Visualisierung war, deshalb erhielt der Alchemist das Upgrade zum Sniper mit Holzbein.

Character Art at its Best

Lucas Reiner
ist 2D und Concept Artist bei Mimimi Productions.

/lucas.reiner.92

Yukis Portrait war das erste seiner Art und entstand in meiner Freizeit. Als Charakter gab es sie zwar schon, aber noch nicht wirklich detailliert, auch ihr Blick war noch weit nach unten gerichtet. Später habe ich den Pinselstil hinzugefügt und die Farben nach unten immer weiter verlaufen lassen. Mit Mugen verlief (Anm.: Pun intended) es ähnlich. Nachdem ich die Entwürfe über mehrere Tage bearbeitet hatte, nahm ich sie mit ins Studio und hab sie vorgezeigt. Da hieß es nur »Cool – jetzt die anderen!«

Die Protagonisten

In **Yukis** Fall musste ich weiter nach vorne schauen lassen, da die Charaktere auf den Portraits im Spiel erkennbar sein mussten. Als ich mit ihrem Bild fertig war, sagte das Team quasi einstimmig, dass Yuki aussähe wie meine Freundin – allerdings hab ich sie nie als Vorlage genommen, das geschah völlig unterbewusst. Tja, damit war meine Freundin also das inoffizielle Model für Yuki.

Für **Aiko** hatte ich am wenigsten Referenzen, aber ihr Gesicht ist ohnehin die meiste Zeit verdeckt; in ihrem Fall passte das aber auch zum Charakter.

Takuma ist inspiriert von Mister Miyagi und The End aus »Metal Gear Solid 3 – Snake Eater«. Der Bart, die grüne Kleidung, aber eben der Marderhund Waschbären anstelle des Papageien. Das war nie unsere Ursprungsidee, doch auch Hambo ist riesiger Metal-Gear-Fan. Wir haben es nie darauf angelegt, aber irgendwann kamen wir zu dem Schluss »Hey, also eigentlich ist das ja The End ...!«

Für Ninja **Hayato** gab es mehrere Einflüsse, darunter ebenfalls Metal Gear, oder auch Charakter-Klassiker wie Hanzo aus »Samurai Showdown«. Im Fall von **Mugens** Gesicht hab ich mich ganz klar an Ken Watanabe (»The Last Samurai«) orientiert.

Die für mich größte Herausforderung war tatsächlich der Pinselstil, denn eigentlich bin ich eher ein Fan klarer Linien, wie man sie in Comics findet. Ich mag den Stil sehr gerne, allerdings habe ich damit ehrlicherweise ein wenig gekämpft.

Hayato ist kräftig und im Töten geübt, daher benötigt er für einen Nahkampf-Kill gerade einmal 2 Sekunden.

Wilde Animationen

Als sich Gameplay und Look immer weiter festigten, bekam jeder im Team langsam auch ein besseres Gefühl für die Charaktere. Wir diskutierten oft darüber, was zu bestimmten Figuren passt und welchen Style sie haben. Das passierte größtenteils basierend auf ihren Archetypen, denn Story und Persönlichkeiten standen zu diesem Zeitpunkt noch nicht fest.

Ein gutes Beispiel dafür ist erneut unsere junge Diebin und ihre Nahkampf-Animation: Von ihr wussten wir, dass sie als Straßenkind aufgewachsen war und dass sie gelernt hatte, körperliche Schwäche mit Geschick und Wildheit auszugleichen. Der erste Vorschlag für ihren Nahkampf-Kill sah daher so aus, dass sie ihrem Gegner auf die Schultern springt und ihn mit ihren Beinen erwürgt. Die Idee dahinter war, dass es leise und langsam geschehen sollte. Wir merkten allerdings, dass viele im Team mit diesem Konzept unzufrieden waren, denn es passte einfach nicht richtig zu dem kleinen wilden Mädchen. Dann kamen wir auf die Idee, ihr einen kleinen Dolch zu geben. Aufgrund ihrer kleinen Statur, müsste sie dem Gegner zuerst ins Bein stechen und ihm dann schnell das Messer schnell in den Hals zu rammen, damit er nicht schreien konnte. Zugegeben, ziemlich brutal, aber jeder spürte sofort, wie gut ihre wilde Seite nun zum Vorschein kam. Von da an wussten wir auch, dass unsere Yuki einen kleinen Knacks hat.

Die Stimme Finden

Im Zusammenspiel mit Art und Game Design formten sich die Charaktere schließlich soweit, dass die ersten Dialoge geschrieben werden konnten. Da es noch keine richtige Story gab, ging es primär darum, für jeden eine Stimme zu finden und natürlich hatten wir auch hier unterschiedlichste Vorstellungen. Lange Zeit war Ninja Hayato ein raubeiniger und humorloser Charakter, dem es nur um die Mission ging. Auch Schütze Takuma war in früheren Iterationen noch sehr mürrisch und verbittert, die Hälfte seines Textes bestand aus Gebrummel und einsilbigen Antworten. Später, als wir uns sicherer im Ton waren, wuchs der Drang nach möglichst farbenfrohen Charakteren. Hayato mauserte sich zum angeberischen Draufgänger, der Sniper zum lebensfrohen Kauz. Der Grundton blieb ernst, war am Ende aber um eine gute Portion Menschlichkeit reicher.

Bis wir die Stimmen unserer Charaktere gefunden hatten, besprachen wir regelmäßig jede geschriebene Zeile. Anschließend ging es darum auszuarbeiten, wie die Figuren zueinander stehen, wieviel man von ihnen mitbekommt, wie man sie kennenlernt, usw. Hierfür orientierten wir uns daran, welche Charaktere in Sachen Gameplay gut miteinander funktionierten, in welchen Konstellationen sie vorkamen und in welche Situationen unsere Missionen sie bringen würden.

Schleichen und Reden

Als es darum ging, während des Spiels die Story zu erzählen, war uns klar, dass wir den Spielfluss so wenig wie möglich unterbrechen durften. Stattdessen wollten wir unsere Charaktere während einer Mission häufig miteinander reden lassen. Dadurch, dass unsere Missionen sehr offen gestaltet waren, ergab sich das Problem, dass wir meist nicht vorhersagen konnten, wo der Spieler die Charaktere positionieren oder auf welchem Weg er die Missionsziele erfüllen würde. Unsere Lösung bestand darin, dass wir uns auf kurze Dialoge fokussierten, die sich aus den Aktionen des Spielers ergaben. Die Charaktere kommentieren ihre Aktionen gegenseitig und gehen so indirekt auf das ein, was der Spieler macht. Auf diese Weise stärken wir die Beziehungen zwischen den Teammitgliedern auf eine natürliche Art und erzuegen zusätzlich den Eindruck, dass sie die Mission gemeinsam lösen. Ein Beispiel: In der zweiten Mission stolpert unser Ninja erstmals über die junge Diebin. In einer Cutscene zu Beginn wird unmissverständlich klar, dass die beiden aufeinander angewiesen sind, obwohl der Assassine lieber alleine arbeiten würde. Der Rest der Mission

Die ikonischen Charakterportraits der fünf Assassinen stammen aus der Feder (oder vielmehr dem Pinsel) von Concept Artist Lucas Rainer.

verläuft ohne Zwischensequenz. Im weiteren Verlauf des Levels lernt der Spieler die Skills der Diebin kennen, während Hayato deren Einsatz kommentiert. Gleichzei-tig reagiert die Diebin auf sämtliche Aktionen des Ninjas. Je länger man spielt, desto mehr bekommt man das Gefühl, dass sich die beiden immer besser verstehen und respektieren, bis Hayato letztlich sogar Yukis Aufnahme ins Team befürwortet. An Stellen wie diesen funktioniert das Story-Writing am besten, aber natürlich haben wir auch ein paar Negativbeispiele, aus denen wir einiges lernen konnten – etwa wenn Yuki und der Scharfschütze ein ruhiges Gespräch inmitten einer tobenden Schlacht führen. Da die Story recht spät finalisiert wurde, konnten wir auf Cutscenes nicht ganz verzichten. Trotzdem tragen diese Gespräche zu großen Teilen dazu bei, dass unsere Charaktere so gut bei den Spielern ankommen.

Englischer oder japanischer Tee?

Die letzten Monate der Entwicklung waren dominiert von Script-Revisionen. Für ein Projekt dieser Größe hatten wir vergleichsweise wenige Lines, daher musste jede einzelne umso ausdrucksstärker sein. Dann ging es ans Vertonen, mit Voice Acting als weiterem Neuland für das Team. Dem Tonstudio OMUK hatten wir vorab Character-Sheets und Beispieltexte geschickt, anhand derer wir die Sprecher auswählten. Anschließend ging es nach London, wo wir eng mit Voice-Directors und Actors im Studio zusammenarbeiteten. Direkt vor Ort zu sein, entpuppte sich als extrem wertvoll, da wir so gemeinsam letzte Anpassungen im Skript machen und den Charakteren auf diese Weise den letzten Schliff geben konnten. Außerdem war es immer hilfreich, den aktuellen Build des Spiels als Referenz zeigen und die gepeinigten Sprecher durch den ein oder anderen komplizierten japanischen Namen lotsen zu können.

Welcher ist dein Favorit?

Nach Release waren wir natürlich super glücklich dass das Spiel so gut ankam. Viele Leute haben unsere Charaktere ins Herz geschlossen, auf Youtube entdeckten wir sogar Anwendungen für Skills, auf die wir selbst nicht gekommen wären. Im Steam-Forum konnten wir Diskussionen verfolgen, welcher Charakter der beste sei. Was uns am meisten freut? Nun, dass es keine klare Antwort gibt und jeder einen anderen Favorit hat.

Rückblickend lässt daher sagen, dass unsere Herangehensweise, die Charaktere aus dem Gameplay heraus zu entwickeln, die richtige war. Für unser nächstes Projekt in diesem Genre nehmen wir uns vor, die Story mehr auf Charakter-Level zu halten und die einzelnen Momente im Gameplay noch stärker auf Ton und Setting der Missionen anzupassen.

Martin Hamberger

Bezaubernd tödliche Animationen

Cem Erdalan is 3D Artist at Mimimi Productions.

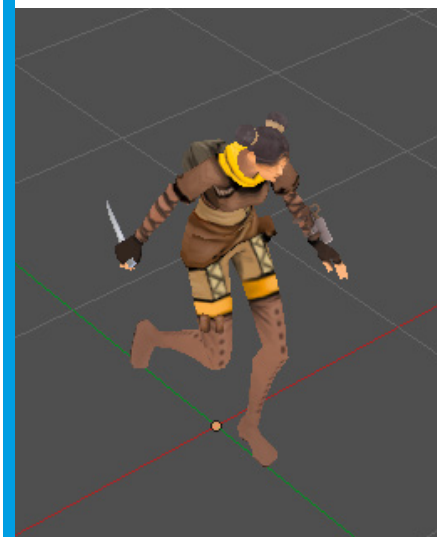
@cemtleman

Jeder Protagonist hat seine eigene Story, Persönlichkeit und Fertigkeiten, und das wollten wir auch im Erscheinungsbild darstellen, über die Silhouette, Statur, Haltung und Animationen. Unsere Diebin Yuki etwa hat bereits in frühem Kindesalter ihre Eltern verloren und ist, auf sich alleine gestellt, auf der Straße aufgewachsen etwa ist. Das Stehlen lernte sie von Straßendieben, sie ist sehr agil und meidet den direkten Kampf, schleicht sich also eher an ihre Gegner heran, um ihnen kurzerhand die Kehle durchzuschneiden. Sie bewegt sich nach vorn über gebeugt, den Dolch immer in greifbarer Nähe.

Der Animationsstil

In Shadow Tactics liegt der Fokus klar auf Gameplay, dem untergeordnet waren daher Kill-Animationen und Charakter-Movement. Die Animationen erfüllen in erster Linie ihren Zweck, sind simpel, klar lesbar und passen zur Persönlichkeit des jeweiligen Charakters. Detailliertere Animationen, z. B. das Bewegen einzelner Finger, waren unnötig, da man sie bei dieser Kameraentfernung ohnehin nicht wahrnehmen würde.

In Spielen wie »Diablo«, »Dota« und »League of Legends«, um nur ein paar Beispiele zu nennen, sind die Charaktere oft »over-animated«, also sehr übertrieben animiert. In Shadow Tactics wurden die Animationen eher schlicht, natürlich und vor allem realistisch.



Animationshilfen

Internetreferenzen

Die ergiebigste Quelle für Referenzen ist natürlich das Internet. Für die Gestaltung der Schwimmanimationen unserer Hauptcharaktere habe ich mir Videos von olympischen Schwimmern angesehen, die ich sogar in den Hintergrund von Blender laden und schlicht nachanimieren konnte. So bin auch bei der Laufanimation unseres Ochsen verfahren.

„Den Denzel machen“

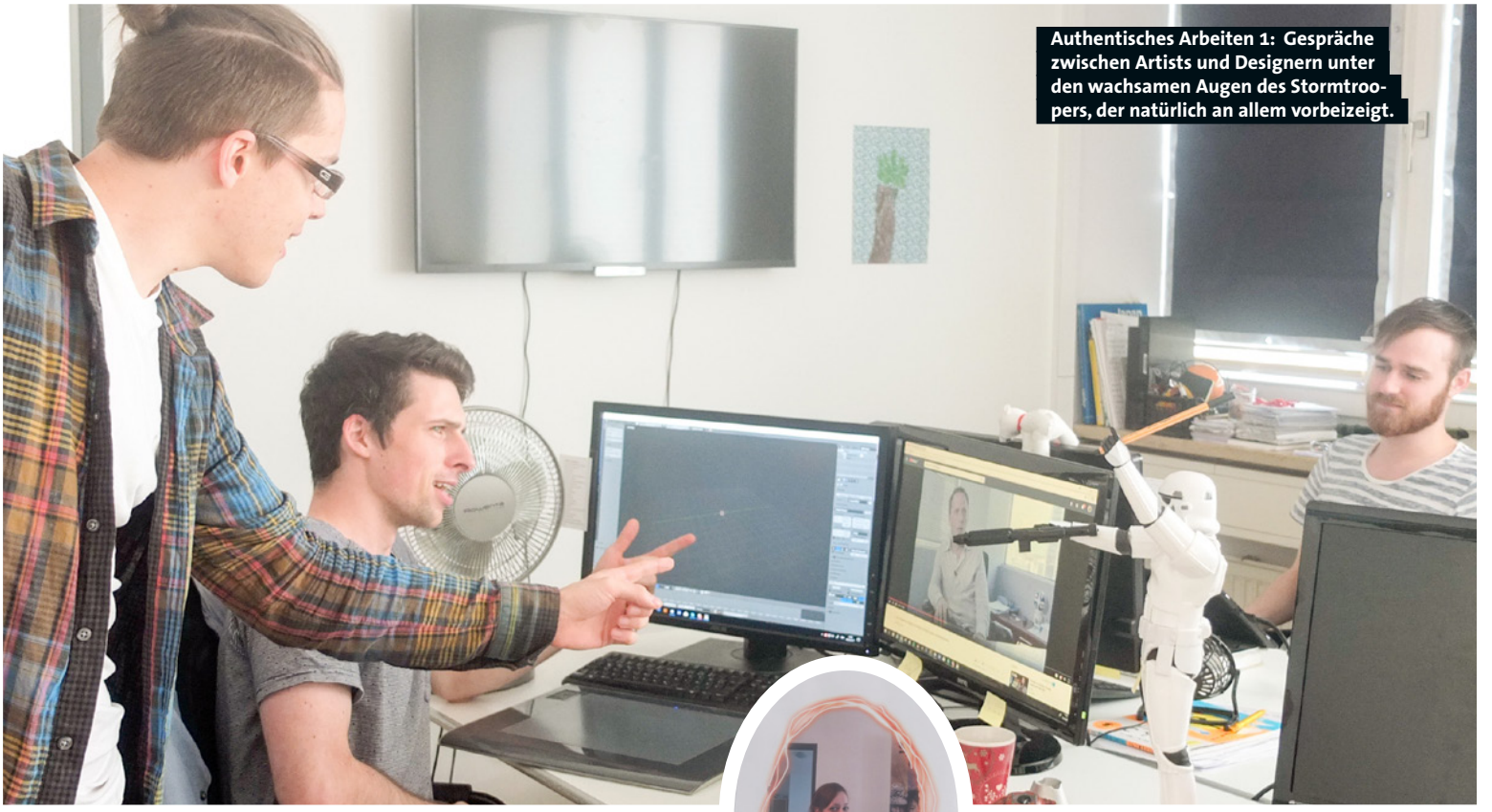
Die Schauspielerei hilft mir als Animator sehr oft, um mich in eine Figur hineinzuversetzen, um ein Gefühl für sie zu bekommen. Ab und an nehme ich mich dabei sogar auf, aber das zeig ich lieber nicht! Wir haben auch oft als Gruppe eine Situation durchgespielt. Ich erinnere mich noch sehr gut an das Meeting mit den Game Designern und Artists. Da haben wir uns unter anderem die Frage gestellt, wie Yuki etwa eine Wache angreifen und töten würde, die einen Kopf größer und stärker ist als sie. Einer von uns mimte die Wache, die anderen standen drumherum und haben über Attacken nachgedacht. Wir waren uns relativ einig, dass sie sich geduckt von hinten anschleicht, um ihr Opfer schnell und effektiv zu überwältigen.

Das Ergebnis ist im fertigem Spiel zu sehen: Yuki schleicht sich an, zückt den Dolch und schlitzt der Wache mit dem ersten Schwung die Kniekehlen auf. Dadurch sackt das Opfer zu Boden und findet mit einem tödlichen Stich in den Hals ein Ende.

Gameplay & Art „Hand in Hand“

Als Animator hatte ich gewisse Vorgaben von unseren Game Designern. Die Kill-Animationen, um bei dem Beispiel von eben zu bleiben, sollten unterschiedliche lang sein. Während das unerfahrene Straßenkind Yuki sich erst einen Vorteil verschaffen muss und ca. 3 Sekunden für einen »Kill«benötigt, bringt der erfahrene Ninja Hayato seinen Gegenspieler in gerade einmal 2 Sekunden zur Strecke. Aus diesem Grund haben wir im Entwicklungsprozess oft zunächst nur Prototyp-Animationen erstellt und von unseren Codern einbauen lassen. So konnten wir gut abschätzen, ob das Feature Spaß macht, bo die Timings im Rahmen des Gameplays stimmen und ob die Animation gut zu unserem Charakter passt.

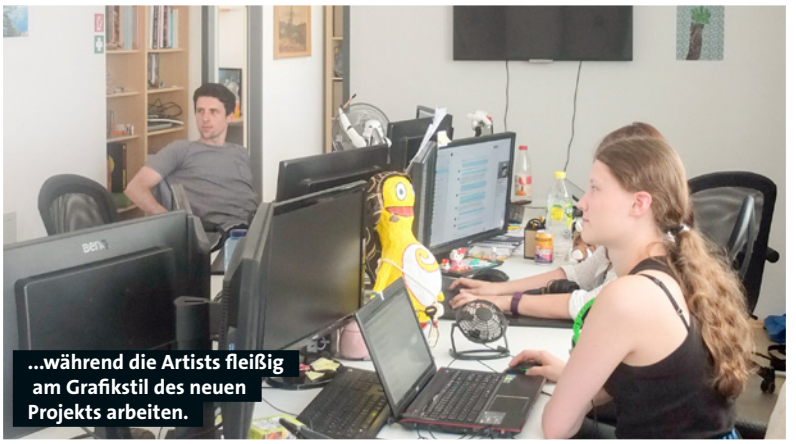
Ein Tag bei ... MIMIMI PRODUCTIONS



Authentisches Arbeiten 1: Gespräche zwischen Artists und Designern unter den wachsamen Augen des Stormtroopers, der natürlich an allem vorbeizeigt.

Wichtig für die frustrierenden Momente bei der Spieleentwicklung: der Boxsack im Aufenthaltsraum.

Authentisches Arbeiten 2: Hambo posiert völlig natürlich im Büro der Designer,...

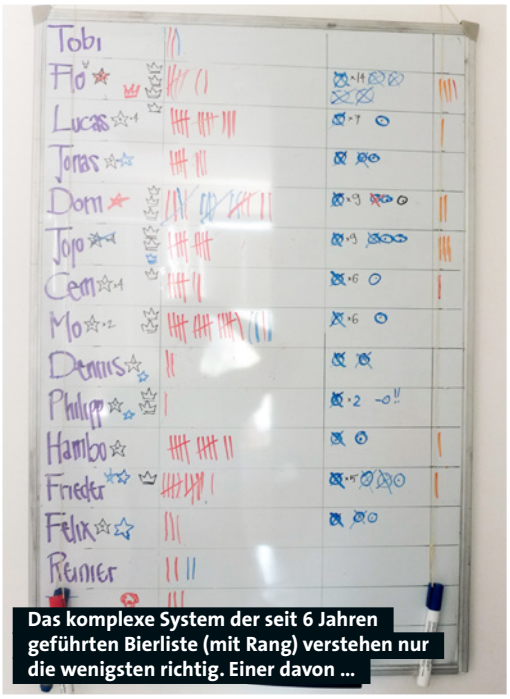


...während die Artists fleißig am Grafikstil des neuen Projekts arbeiten.

Der mächtige »Shadow Tactics« Marketingplan hängt inzwischen in der zweckentfremdeten zweiten Küche.

Wenn man eh schon in München vorbeischaut, um Mimimi Productions in ihrem natürlichen Lebensraum zu beobachten, dann kann man auch gleich ein paar Fotos schießen und die Entwickler vom wohlverdienten Urlaub abhalten.


Anatidaephobie (die irrationale Angst von einer Ente beobachtet zu werden) in Perfektion



Das komplexe System der seit 6 Jahren geführten Bierliste (mit Rang) verstehen nur die wenigsten richtig. Einer davon ...


... ist der Zweitplatzierte: Geschäftsführer Johannes Roth.


Wilderer am Werk: Schießstandtrohäen vom Oktoberfest und Kühbacher Brauerreifest.


Award-Wand: Im Meetingraums sind alle bisherigen Projekte und Preise verewigt.


Während im Coder-Raum heilige Zeilen in Visual Studio verewigt werden,...


... hält Studiomanagerin Anna allen den Rücken frei.


Die (noch) ungerahmte Fan Art hängt im Raum der Artists.


Authentisches Arbeiten 3: unser Marketing-Genie Dennis.