Authoring tool descriptors[1]

Our 38 descriptors are divided into two groupings: contextual descriptors, and user-facing descriptors.

Contextual descriptors. (Descriptor groups A and B in the online resource [1]) include relevant characteristics that are extrinsic to a tool's functional design: a tool's popularity, ownership model, exterior resources that relate to it, etc. In other words, these are 'paratextual' descriptors that map some central aspects of the context and impact of a tool. This is more than merely a meta-analysis: a full understanding of a tool requires addressing the broader context and conditions of its creation and use [50]. Contextual descriptors thus capture the financial and discursive environment of actual tool use.

Group A of descriptors consists of essential, defining and mostly self-explanatory characteristics, including (1) *Name*, (2) *Creator & affiliation*, (3) *Year of release*, and (4) *Category* (see 4.2 above). Two other descriptors are (5) *End-product media type* –-values include real-time graphics\text\video\AR\VR (fettered\unfettered)\hybrid; and (6) *Main target audience*, values for which include youth/students, amateurs/enthusiasts, storytellers (no programming background), programmers (no creative background), professionals or semi-professionals (some background in both), mixed.

Group B of descriptors are those that attempt to capture the context of a tool: usage metrics, links to specific resources and business aspects. Usage metrics include the (7) *Vitality* of a tool – tools that are 'alive' are in current creative use (and in the vast majority of cases, maintenance), tools 'in limbo' can still theoretically function but are barely or not at all in current use, and 'dead' tools are not in use and have also become impossible or very hard to access, run and\or work with. (8) *Number-range of products made* attempts to capture popularity over time, with ranges of 0-100/100-500/500-1000/1,000-10,000\10,000-50,000\50,000+. From an academic perspective, following the links allows us to glean into the culture and discourse that surrounds a tool: the ways in which a tool presents itself, works published with it are accessed, and the process of learning it is facilitated, all strongly affect the tool's character without being directly ingrained in its platform. Access to longer pieces of textual analysis and critique of the tool (including, when available, academic papers) provides a possibility for more abstract or in-depth reflection. Related online resources descriptors include links to the (9) *Homepage*, (10) *Publishing portal(s)*, (11) *Textual*

[1] This document was compiled in 2018 by Yotam Shibolet, an MA student at the department of Media Studies at the university of Utrecht, as part of a research internship at the Professorship Interactive Narrative Design. The internship was supervised by Prof. Hartmut Koenitz and Drs. Noam knoller. The project website is currently at interactivenarrativedesign.org/authoringtools/

analysis source(s), (12) *Sample product(s)* and (13) *Tutorial(s)*. Finally, four descriptors capture the business model: (14) *Ownership type*, (15), *Latest stable release version* (number + date), (16) *Cost* and (17) *License type.* The way a tool's owner(s) moderate, facilitate and manage its use and development dramatically shapes the type of access and experience prospective authors are given.

User-facing descriptors. (descriptor groups C and D in the online resource [1]) describe the tool's intrinsic structures and are evidenced or experienced through the processes of authoring, as facilitated through the tool's interface. In other words, these descriptors analyse the tool itself: the structure, affordances and interface of its work environment, and the range of end-products they allow authors to produce.

The six descriptors in group C describe a tools' technical structure and functionality: the 2 descriptors (18) *Programming language written in* as well as the (19) *Programming\scripting language employed in work-process (user-facing)* describe the way the tool uses code, whereas (20) *Role of coding in the creative process* tells us how integral programming is to the authoring process (primary\ central\ optional\ supplementary\ none), and thus to what extent a prospective author would needs to know, or to be willing to learn, the programming language employed in the work process in order to be able to create effectively. The tool's *(21) Work platform(s)* (Windows\MAC\LINUX; PC/browser-based) is captured, as well as the tool's interoperability, represented by (22) *Import formats* and (23) *Export formats*.

Group D of 15 descriptors describes the details of the author-facing design process, from interface to the types of artefacts that can be created, and the sophistication of the cognitive scaffolding [27] structure provided to the author by the tool. Without conducting user studies on a tool, describing such a process requires observation of its affordances, and a somewhat subjective speculation about how various functionalities of a tool would actually be used by an author.

Six descriptors define the tool's author-facing interface. The online resource accommodates (24) *Interface screenshot(s)* as well as entries for a tool's (25) *Primary design unit(s)* and (26) *Main design window(s)*, which together point out which functionalities are highlighted in the tool's procedural rhetoric. The (27) *Work environment UI model* describes the interface metaphor: is it based on a timeline, a lexia choice-space emphasising links between units; multiple windows onto a 3D space, some other option (such as the multiple-choice menus that dictate potential links in Hypedyn [51]) or any combination thereof.

(28) *UI modelling Type and level*, is one of the more complex descriptors in this framework. It attempts to classify the types of actions an authoring tool requires from authors. These can vary greatly from writing abstract code and granular graphics to tinkering

with pre-set settings. This variety is expressed by the UI's level of modelling, which we have reduced to 6 paradigms: *Visual Templates* (readymade structures, which the author calibrates and integrates together), *Extendable objects* (basic modular units whose writing can directly create more units of objects by directly linking to them and drive the authoring process), *drag & drop* (an object-based interface where different kinds of units are placed directly inside the workspace), *visual coding, text menus, and abstract coding (could be templated)*. These paradigms are adapted from an existing division into 4 different types of "game engine features", by level of abstraction, in [52]. IDN tools, despite not all being game engines, can be divided similarly, with the addition of some further categories. Tools with higher levels of modelling offer a simpler, but less adaptable and customisable process – but this may still vary to high extents within a given paradigm (simplified visual coding tools such as Game Salad [53] are simpler to program than Unity's [28] drag-and-drop interface, for example). Naturally, tools often combine multiple paradigms for performing different actions. (29) *Design interface intuitiveness* describes whether a tool's UI is easy or tough to figure out. Influenced by the other descriptors in this group, it is determined primarily by the phenomenological quality of 'good design' – key features are prevalent in the interface screen, the appearance and placing of things coheres with their function, and the program generally feels friendly and seems to nudge the user towards appropriately creative use. Two descriptors serve to gauge the complexity of the authoring process in correlation to its abilities and the efficiency of its design. (30) *Initial learning curve complexity correlation* accounts for the experience of first time use and is influenced by a tool's end-product media type, central affordances, GUI modelling, and intuitiveness of interface design. (31) *Advanced authoring complexity correlation* is influenced, rather, by a tool's depth and wealth of features, as well as by the diversity of possible end products. Both are necessarily subjective measures at this point, until intersubjective measures can be established through further research.

Finally, seven descriptors attempt to describe the more advanced affordances, those most unique to IDN: narrativity, procedurality and (embodied) interaction.

Two descriptors relate to the way a tool represents narrative. While all the tools in our online resource may be treated as IDN authoring tools, their (32) *Degree of emphasis on narrative structuring* - the ways in which they manifest the narrative design possibilities they provide – varies greatly. Some multi-purpose tools, particularly game engines, essentially do not highlight narrative design to any extent. they can offer many features of great efficacy to narrative design, but authors wishing to use them to create an IDN work may still have to find their own way to represent narratological constructs, typically through additional coding.

Other tools explicitly treat narrative design as their central aim, but still represent narrative differently from others, and this is registered in the descriptor (33) *Prevalent narrative elements & concepts* - the number and types of narrative concepts they employ. Inclusion or exclusion of a narratological concept, such as ‘character’, ‘events’, ‘goals’ or ‘drama’, underscore the design process in widely different ways by presenting the author with different building blocks that affect the author’s focus and implicit understanding of what narrative means, or what dimensions are centrally important to a narrative work.

Similar to narrative, procedurality accounts for elements of authoring that a tool can generate algorithmically. Procedural elements include drama or event managers and\or various variables that affect the course of an instantiated IDN, as well as character behaviour, environment generation, conditional global counters, combat systems, randomization of elements such as appearance and stats of items, etc. (34) *Prevalence of procedural elements* marks their centrality in the tools intended authoring process (primary\central\optional\supplementary\none), while (35) *Main available procedural elements* lists the primary procedural actions and categories available. It should be noted that inclusion of procedural elements does not necessarily require the author to code, and they can just as easily be employed through readymade templates or drag-and-drop objects (see descriptor 28 above).

Finally, Interaction model design accounts for elements that determine the affordances and design space of the interaction model of works designed by an authoring tool – the UI hardware and software through which the end-user interacts with the storyworld, and then the specific meaning assigned in the work to those UI elements and to the ways users perform the work through them [25]. These design descriptors partly correspond with an end product media type, exportable formats, and UI model of work environment, to the extent that these are standardised. (36) *End Product platform & interface* accounts for the basic range of platforms (PC\Playstation\XBOX etc.) and interface paradigms (mouse clicks, mouse & keyboard, joystick, motion controller, gestural capture etc.) that a tool’s end-products can deploy. It also tells us whether or not the tool supports multiplayer functionalities (which is often correlated with the above range) – which requires a somewhat different range of interface, mechanics and storyworld design. Delving somewhat deeper, (37) *Available end-product interaction model(s) & degree of flexibility* tells us which specific interaction models with the storyworld the tool is prefigured to produce, and to what extent it is flexible enough to afford customised interaction models. This is affected by both a tool’s technical affordances and the design of its own interface, which can typically slant towards always producing works with the same specific interaction model. For example, many game engines can technically produce a text adventure, but nothing in their structures encourages anyone to do so if they're not already dedicated to this specific genre. (38) *Additional key interaction design affordances*, the last descriptor, accounts for any important design components or areas that define and structure an end-product's interaction model. This is in addition to any procedural elements (such as camera movement and character UI) capable of playing a role in this design.

In conclusion, it should be noted that key affordances relating to procedurality, narrative, and interaction model may or may not be congruent: for example, a procedural capacity to generate background vegetation does not double as an affordance for either narrative or interactivity, while a procedural capacity for generation of dramatic events relates to all three types simultaneously. It is our working hypothesis that a degree of such congruence, or at least a degree of structural inclination towards deploying these affordances in a cohesive manner, contributes to an authoring tool's likelihood of producing powerful IDN works [25, 54]. As narrative and interactivity should be deployed in cohesion rather than as two competing or contradictory qualities [27], [55, 56], a hypothetical game engine that only directs the author to think of narrative when constructing cut-scenes that break away from the interaction model, would be inferior to one that would channel narrative occurrences as a dynamic part of being in, and interacting with, the storyworld.

Tool Comparison

Choice of tools for comparison

We have selected 3 authoring tools for a table of comparison using our parameters for IDN authoring tool analysis: Unity, Twine and ASAPS. These 3 tools were chosen for several reasons:

- They provide a cross-section of all 3 main ontological categories (game engine, IF, hybrid), as well as of the 3 main different ownership\tool development models: company-run (Unity), academic (ASAPS), and community-run passion project (Twine). (Unity and Twine are also open source). They also differ and vary in other fundamental ways which the comparison will show. Though our framework also aims to be capable of comparing the nuances of authoring tools of very similar natures, this comparison highlights its (likely more unique) ability to cross-examine IDN authoring tools of entirely different types and paradigms through the same set of parameters.
- They are currently among the most successful and influential tools in their categories. Unity and Twine are the most used tools in their categories in the recent decade by a wide margin. Though ASAPS' scope of products is relatively low, both its longevity

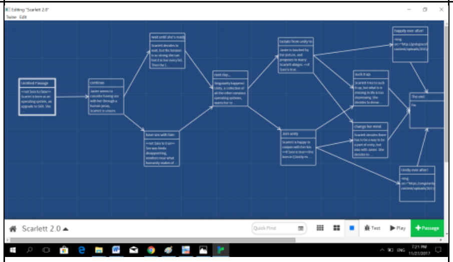
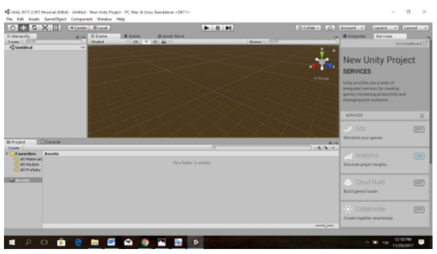
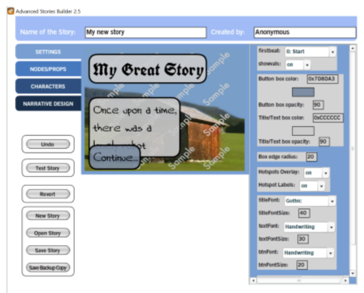
and scope of products are impressive for an academic project of limited public availability.

- They are highly acclaimed tools that have been subject to much study: Unity, like all game engines, has not been extensively written about in the IDN context, but the amount of academic work on it is immense, as befits a tool whose community-of-users outnumbers any of its predecessors, to our best estimation. Twine is often mentioned in papers on IF and has been the subject of a dedicated platform-studies analysis. ASAPS has been the subject of much follow-up work by Koenitz, including the only meta-analysis of an authoring tool's end-products that we know of to date. Carey's recent PhD dissertation on writing strategies for interactive writers notes that "Although there has been extensive development in authoring programs over the past 20 years, there are very few that come close to matching Murray's proposed software. Two programs stand out in this regard: Twine and ASAPS" (Carey 2018, 21).

Comparison table

Parameter	**Twine**	**Unity**	**ASAPS\Advanced Story Builder (ASB) Engine**
Creator & affiliation	Chris Kilmas, independent American game designer	Unity Technologies (funded in Denmark, 2004)	Hartmut Koenitz, academic (initiated at the Advanced Stories Group at Georgia Tech, established by Koenitz, 2005)
Year of release	2009 (Twine 2 released 2014)	2005	2011 (first academic publication, in development since 2007)
End-product media type(s)	Link-based Interactive textual fiction	2D\3D games + VR\AR\MR	Visual-novel-like text + animation hybrid
Main target audience	Amateur IF enthusiast, independent artists	Amateur, semi-pro and professional game designers	Narrative design students
'Life' status	Alive	Alive	Alive (advanced beta prototype in active classroom use, available upon request)
Rough scope of products made	1,000-10,000	50,000+ (over 300,000)	150 +- (more than 60 projects listed in Koenitz 2015 overivew, current number is assumed at about 100)
Homepage	https://twinery.org/	https://unity3d.com/	http://advancedstories.net/
Publishing portal link(s)\overview of products made	http://ifdb.tads.org/search?searchfor=system%3Atwine&searchgo=Search+Games https://twinery.org/	Amateur games platform: http://www.y8.com/tags/unity3d List of published commercial games: https://en.wikipedia.org/wiki/List_of_Unity_games Youtube video of top 10 Unity games:	http://advancedstories.net/?cat=50 Koenitz, Hartmut, and Kun-Ju Chen. "Genres, structures and strategies in interactive digital narratives–analyzing a body of works created in ASAPS." International Conference on Interactive Digital Storytelling.

		https://www.youtube.com/watch?v=oCxv0twTww4	Springer, Berlin, Heidelberg, 2012. https://goo.gl/S5ucn3
Textual analysis sources	‘Anna Anthropy manifesto (see ‘Why Twine’) http://nightmaremode.thegamerstrust.com/2012/11/25/creation-under-capitalism/ Friedhoff, Jane. "Untangling Twine: A Platform Study." DiGRA Conference. 2013. S	Messaoudi, Farouk, Gwendal Simon, and Adlen Ksentini. "Dissecting games engines: The case of Unity3D." *Network and Systems Support for Games (NetGames), 2015 International Workshop on*. IEEE, 2015. Prazina, Irfan, and S. Rizvic. "Unity hyperlinked interactive digital storytelling." *Proceedings of Central European Seminar on Computer Graphics CESCG*. 2015.	Koenitz, Hartmut. "Extensible tools for practical experiments in idn: the advanced stories authoring and presentation system." International Conference on Interactive Digital Storytelling. Springer, Berlin, Heidelberg, 2011. Koenitz, Hartmut, and Kun-Ju Chen. "Genres, structures and strategies in interactive digital narratives–analyzing a body of works created in ASAPS." International Conference on Interactive Digital Storytelling. Springer, Berlin, Heidelberg, 2012.
2 Sample end-products	*Howling Dogs* (Porpentine, 2012) http://slimedaughter.com/games/twine/howlingdogs/ *Sacrilege* (Cara Ellison, 2013) https://unwinnable.com/wp-content/uploads/2013/04/Sacrilege.html	*Ori and the Blind Forest* (Moon Studios, 2015) https://www.youtube.com/watch?v=cklw-Yu3moE *Super Hot* (Piotr Iwanicki, 2016) *http://www.y8.com/games/super_hot*	*Imprisoned* (Unmentioned author, 2013) *http://advancedstories.net/?p=121* *Breaking Points* (Hartmut Koenitz, 2014) http://hartmutkoenitz.com/Work.html
Tutorial link(s)\introduction\wiki links	https://twinery.org/wiki/ http://twinery.org/cookbook/ Tutorial playlist by Dan Cox: https://www.youtube.com/playlist?list=PLlXuD3kyVEr5tlic4SRe6ZG-R9OyS1T4d	https://unity3d.com/learn/tutorials Official Unity 2D game development walkthrough: https://www.youtube.com/watch?v=4qE8cuHI93c	‘Hartmut Koenitz Introduces his ASAPS’: https://www.youtube.com/watch?v=-L6hDcW7Fj4&t=1s http://advancedstories.net/?page_id=465 (Additional tutorial text file available upon request)
Ownership type	Open-source; community-run	Open-source; company run	Academic project; privately owned
Most recent stable release version	2.21 (January 2018)	2017.4 (March 2018)	Beta 18 (iOS playback engine is currently in late stage development)
Cost (funding\business model)	Free	Free; costumers pay for assets + royalties for successful games?)	NA (not publically available, given by demand)
Author user license type (allows self-publishing of end-products?)	GPL v3; Yes	Proprietary; Yes (authors allowed to sell games royalty-free, selling assets is more complex)	MIT (get exact info from Hartmut!)
Work platform(s)	Linux, Mac OS X, Windows, Web application	MAC OS X 10.9+, Windows, iOS, Android	Mac OS X, Windows
Programming language(s) written in	Javascript (V1 in Python)	C++ (Runtime); C# (Unity API)	ActionScript\Haxe
Programming language	Twee Harlowe\Sugarcube\Snowman	Unicode (custom-programming language)	ASML (XML-based markup language)

employed in work-process			
Role of code in the creative process	Optional (basic branching story can be written without code, basic coding implementations are required for counters, conditionals, etc.)	Optional-central (advanced features only, but there are many possible ones that the tool integrates)	None-supplementary (supports direct ASML coding\debugging via Sublime) (though ASML additionally aims to function as middleware, and is beneficial to learn for better understanding of the tool) ASML = both middlemware and programmable by Sublime
Importable formats	Full HTML compatibility Other media: Twine 1: Images: PNG, GIF, JPEG, WebP and SVG Twine 2: none All media formats can be ingrained into a Twine story via HTML, but this is quite clumsy and requires more coding knowledge than any other simple Twine function	Images: BMP, TIF, TGA, JPG, PSD (limited) Sound: MP3, OGG, WMV, AIFF, MOD, IT, S3M, XM 3D model files: max, .blend, .mb, .ma and more Animation: d .fbx, .dae (Collada), .3ds, .dxf, .obj, and .skp Video: anything playable on quicktime (.mov, .mpg, .mpeg, .mp4, .avi, .asf and more) Many other graphic file types such as particles, many other media types are partially readable as textures via various Unity assets	JPEG; SWF; FLV, PNG; SWF (Props\Characters) MP3 (sound, transitioning to OGG?)
Exportable formats	HTML	FBX; .blend; COLLADA;.obj Full list is too long to include, Unity supports export formats for 25 different platforms, including VR, AR, smartphones and all popular game consoles	Multimedia folder (SWF; MP3; PNG); operates in Flash via ASMLEngine, iOS and desktop engines available, HTML5
Interface screenshot			
Main design window(s)	Basic lexia space page is Twine's only window	Scene (abstract view); Game (WYSIWYG); Asset store (buy and import unity assets); Timeline (optional drama system); Inspector; Hierarchy; Animator (Additional windows\workspace structures available in assetstore)	Settings, Nodes\props, Characters, Narrative design, Plot Graph
Primary Design Unit(s)	Passages – unified text lexia unit structure	Gameobject + object child Main types: 3D, 2D, effects, light, audio, video, UI (including text and images), Vuforia (AR and support), camera Components for graphic design Assets for drama and graphical management (scripts, lighting parameters, animation controller, etc)	Beats: multi-purpose elements of 14 types with various functions: info screen, choice screen or variable manipulation Nodes: backdrops for visual beats, can be shared by many different beats. Characters Props
Work-environment UI model	Lexia space	2D\3D WYSIWYG Space design; includes camera navigation in game space	Multiple choice menus; Visual editor; Lexia space (narrative network graph)

UI-modeling abstraction level & type	High Extendable objects (additional lexia are created by customizing existing lexia)	High Drag & Drop based spatial interface, including complex objects such as cameras and effect controls	Low (multiple highly abstract menus for nodes and other narrative functions, default views of narrative beats) High (WYIWYG graphic editor for some beat-types, narrative design graph view presents the story in lexia structure)
Design Interface intuitiveness >HCI/UI	Very high	High-very high	Low
Initial Learning Curve complexity	Very low	Medium-high	Low-Medium
Advanced authoring complexity	Medium	High-very high	High
End product interaction model(s) & degree of design flexibility	Point-and-click text, very low	Any – prefigured towards real-time action & spatial navigation, very high	Point-and-click text\graphical units, low-medium
End-product platform & interface support (enables multiplayer?)	PC (mouse), Smartphone (touchpad); no	Any (PC\mouse and keyboard, consule – joystick and game controller, Kinect, VR, etc); yes	PC (mouse), RFID Can this run on smartphone (touchpad clicks?) ; no
Prevalence of procedural elements\structures	Supplementary-optional	Central	Optional
Main available procedural authoring elements	Conditional linking (if-;else-if); Randomizers, counters (allow procedural linking), 'combat' stats	Game physics, Camera movement, Character AI, lightning scrips, environment generation, etc.	Linking between nodes, randomizers, timed events, condition checks, character states, etc.
Additional interaction design elements	None	Objects interact on collision\trigger\button, PlayerController (movement interface)	Choice interaction (NavigationChoice, ConversationChoice)
Degree of narrative structure emphasis	Medium	Low (high with drama management assets like Fungus)	Very High
Prevalent narrative elements & concepts	Story progression\Events - choice based textual links between lexia Easily integratable support for conditionals, counters, inventory, stats, RPG battles, randomization	None made explicit Scene and possibly timeline screen makes it relatively easy to integrate events into free navigational space structure Tool is flexible enough to potentially fit any sort of narrative elements, and includes plugins for narrative design (such as Fungus), but does nothing in its default interface to condition the author to narrative thought	Specified in main design units: Characters & character state Nodes (conditionals that trigger visual beats) Inventory – props Event screens Beats prefigure the user into multiple narrative aspects and design spaces: Timed events (DurScreen, Set Timer) Counters & global variables (SetCounter, SetGlobal, ConditonCheck)