Tiny Online Game Engines

Jonah Warren
Game Design and Development
Quinnipiac University
Hamden, CT, USA
jonah.warren@quinnipiac.edu

Abstract— **A Tiny Online Game Engine (TOGE) is an online tool that allows designers to create small games of a certain genre quickly and easily, often without programming. This paper explores the current landscape of TOGEs by examining three of the most popular engines currently available online: Twine, PuzzleScript, and Bitsy. Each of these tools streamlines the game development process by defining a strict set of constraints affecting the rulesets, aesthetics, and experiences of the games created with it. The design constraints of each engine are examined along with each engine's interface, its design philosophy, a sampling of games published with it, and information gathered from developers who use it. Included in this study is the metadata from 5,001 games and survey results from 163 game developers. Analyzing these engines from a variety of perspectives may help future tool creators identify strategies for making game development more beginner-friendly, accessible, inclusive, and fun.**

Keywords*— *Game development, Game engine, Accessibility, Constraints, Design philosophy

I. Introduction

Game development can seem overwhelming. Looking at the credits of any successful recent console release begins to reveal the scope of AAA development today. God of War, which was one of the best selling console games of 2018 [14] and sold over 3.1 copies in three days [25], is reported to have had a team of 270 developers working on it for four years [12]. In a 2013 interview, a director at Ubisoft stated that a typical open-world action Ubisoft game can require a team of between 400 and 600 people to develop [18]. A quick perusal of notable game budgets collected by Kotaku in 2014 reveals that a majority of the games listed in the 2010's fall in the $40 to $100 millions dollar range. And it is only getting worse. An analysis of over 250 games released over the last several decades shows that the cost of game development has gone up 10x every ten years [19].

For aspiring designers just starting their journey in game development, things don't look much better. Although the proliferation of freely available professional-grade tools (e.g., Unity, Unreal, Cry Engine) has made it seem like game development has become more accessible, a closer look reveals a challenging environment to navigate. For beginners, these programs can seem extraordinarily complex. A quick search on Unreal online forums for the term "overwhelming" yields pages of posts with titles like: "Feeling Overwhelmed," "Is it me, or is it just a little overwhelming," "I am extremely overwhelmed," and "Beginner's Frustrating Experiences with the Engine." This shouldn't come as a surprise considering that when downloaded, a PDF of the Unreal game engine's documentation is over 8,500 pages and takes up over a gigabyte of space [29]. Even Unity, often considered a simpler engine with a gentler learning curve can often feel intimidating to first time users. A similar search on Unity forums yields similar results with titles like: "Feeling overwhelmed," "Is it just me, or is the Unity community very overwhelming," and "I'm just so overwhelmed." Even its resources for learning can seem monumental. A video series offered on Unity's website covering "the fundamental skills" of game development using the engine contains over 99 hours of content [28].

Over the past five years, a few creative tool creators have created a number of more accessible alternatives, called Tiny Online Game Engines, or TOGEs, each approaching game development from a different perspective. TOGEs offer beginners many possible paths into game development without the friction involved in learning a massive do-it-all game engine. They allow developers to focus on what they are good at and interested in, be it storytelling, world creation, or experimenting with interactivity. They are also useful for more experienced developers who want to explore ideas quickly or make prototypes for larger games. They provide a way for developers to make functional interactive "sketches" of game experiences, as a traditional artist would, trying things and learning from mistakes in preparation for a larger work. This can be especially invaluable in game design where testing game concepts at an early stage is crucial to the design process.

TOGEs also take advantage of modern web technologies, making it possible for developers to develop entire games in a web browser without downloading any software. Although a technological distinction such as this may seem arbitrary, the accessibility this provides shouldn't be underestimated. Having a computer with the required specifications, enough space to install a multi-gigabyte engine, and the internet speed to support that download can be significant barriers to entry. Additionally, most game engines don't support the creation of entire game within it. Other software must be used to create 2d sprites, 3d models, sound effects, and music. Each of these programs comes with a learning curve and price tag. Most TOGEs, on the other hand, provide developers all of the tools necessary to create complete, albeit small, game experiences. This can include browser-based dialog editors, sprite editors, animation tools, audio creation tools, and code editors. These technologies also allow developers to export their creations as HTML, which can be shared instantly and played by anyone with a browser and an internet connection.

So how do TOGEs accomplish all of this? In a word: constraints. TOGEs create a very specific type of game experience within a given genre. As mentioned previously, each TOGE has its own perspective and design philosophy on creating games. The entire design of a TOGE, from its constraints to its interface and documentation, is a manifestation of that philosophy. Aside from making games easier to create from a technological perspective and simplifying the engine's interface, it also serves another purpose: to generate and foster creativity. When designers face limited options, they have the freedom to use those options in less conventional ways—because it's all they've got [20]. Complete creative freedom can often be counterproductive.

Although this study only looks at three TOGE's, many others exist. A few other TOGEs of note include Vertex Meadow, a tool that renders 2D images as explorable 3D terrain [9]; Flickgame, which has been described as the mutant child of MS Paint and Hypercard [21] [8]; Pling Pling, a draw-your-own pinball game engine [22]; and Tiny Choice, a minimalist hypertext game engine [23]. Twine, Bitsy, and PuzzleScript were chosen for this study for a number of reasons. They were the most popular TOGEs used to create games on itch.io (a popular repository for independent games) [10], they all have very active online development communities, and they each have very different design philosophies, providing the opportunity to study a range of perspectives on TOGEs and their design.

II. Twine

A. History

In 2009, Chris Klimas, an interactive fiction writer inspired by Infocom games in the 1980's and ftp.gmd.de (an FTP server hosting interactive fiction) in the 1990's [15], released Twine, a free, downloadable, open-source application for Mac and Windows [11]. Klimas, who had been studying interaction design and experimenting with different ways to create hypertext at the time, describes his motivations as an "attempt to make something that would be friendly to people who were writers more than coders" [13]. In Twine's initial iteration, authors could export their Twine creations as .tws files which, notably, could be exported to HTML and CSS making the games extremely accessible, unlike similar tools available at the time, which required proprietary software to run created games [11]. Although released in 2009, Twine didn't gain popularity until a few years later in 2012 [11]. In Untangling Twine: A Platform Study, Jane Friedhoff attributes Twine's explosion in popularity to a series of blog posts written by Anna Anthropy and her book "Rise of the Videogame Zinesters" [4]. In 2014, Klimas released Twine 2.0 as a browser-based tool. Although downloadable versions of the software are still available, Twine 2.0 gives authors the ability to create Twine games online at twinery.org, save them locally in their browser, and export to HTML by default [30].

B. The Engine

1) The Basics. Twine allows authors to create hypertext. In its most basic form, a Twine game, or story, is text organized into passages that players navigate by clicking on hyperlinks, directing them to other passages. Passages initially appear as white text with blue hyperlinks that quickly fade in on a black page, however, a story's colors and fonts can be easily customized with CSS. Since its publishing format is HTML, Twine games can easily incorporate text effects, sounds, animation, and imagery. Twine also uses Javascript to support the use of variables and conditional logic, enabling designers to incorporate traditional game constructs like inventories, hit points, or scores into their creations.

2) The Interface. After a brief introduction and a few links to orientation materials, an author's first encounter with the Twine interface is the engine's story list: a collection of a

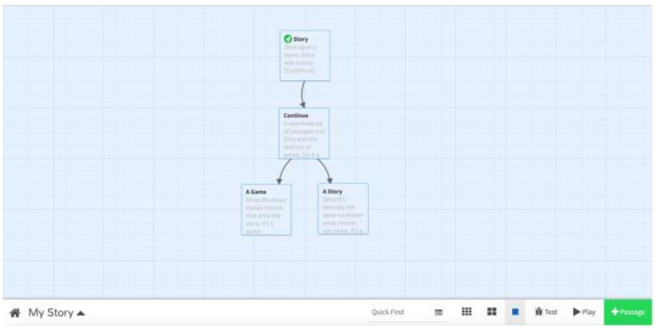


Fig. 1 A screenshot of Twine's story map interface.

developer's previously saved creations. The story list page also allows authors to import existing stories from a file, archive their work, change their language, and select formats. To begin development, authors can either create a new story or click on an existing story from the story list, which takes them to them to its story map.

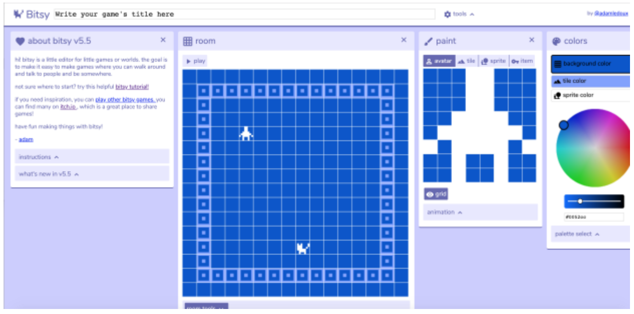
The story map is a canvas that contains all of the passages and connections that make up a Twine story. It's also where the majority of development takes place. By default, the story map for a new story includes a blank passage represented on the canvas by a white box labeled "Untitled Passage." Passages are the core building block of Twine stories. Double-clicking it opens an editor, revealing a passage's structure: a title, tags, and most importantly, a body of text. In addition to adding text to a passage's body, an author can also add hyperlinks by surrounding bits of text with double brackets. For example, if an author entered [[Go west]] in a passage, the text "Go west" would appear as a hyperlink, linking the reader to a passage entitled "Go west." If there is no existing passage named "Go west," Twine will make one. Returning to story map after adding a hyperlink will reveal two passages linked by an arrow. In such a way, the story map interface visualizes a Twine story's structure, displaying all of its passages and connections. Although there's a lot more to Twine (scripting, formats, embedding media), the process of creating passages and making links between them is the core design pattern required to create Twine stories.

3) Design Philosophy. Klimas' stated goal of making a writer-friendly tool for creating hypertext permeates Twine. Everything about it is oriented towards writers, even its vocabulary. Twine developers are "authors" who add to their "story list" by creating "stories," which are made up "passages." This design philosophy also pervades Twine's documentation, which is analyzed in depth by Friedhoff [11]. Aside from a few exceptions (i.e., details on the syntax of formatting text or adding stats), the language used in Twine's documentation is not technical in nature and rarely mentions programming. Instead, it focuses on the interface and the basics of writing hypertext. It also takes time to contextualize interface decisions and the engine's structure. For example, explaining that "because hypertext branches so much, it's easy to get lost," and that a lot of Twine's interface is "dedicated to helping you keep track of your work's structure visually" — a direct reference to its story map interface [27]. The design of Twine's interface is also organized using formats likely familiar to writers. With their draggable, small, white titled squares, passages in story map interface closely emulate sticky notes.

III. Bitsy

A. History

Inspired by the pixelated towns of 2d RPGs like The Legend of Zelda, Adam Le Doux released Bitsy in 2017 [2]. Le Doux wanted to create an engine that freed designers from worrying about “programming details, screen resolutions, platform differences, post-processing effects” and enable them to concentrate on creating a world with a story [1]. The engine’s documentation describes itself as an editor for creating little worlds, with the goal of making it easy to make games “where you can walk around and talk to people and be somewhere.” In a short time period, Bitsy has developed an active following of passionate developers [1].


hi! bitsy is a little editor for little games or worlds. the goal is to make it easy to make games where you can walk around and talk to people and be somewhere.

not sure where to start? try this helpful bitsy tutorial!

if you need inspiration, you can play other bitsy games. you can find many on itch.io, which is a great place to share games!

have fun making things with bitsy!

- adam

Fig. 2 A screenshot of Bitsy’s default panel layout.

B. The Engine

1) The Basics. Bitsy’s graphic language and interactive vocabulary are significantly constrained. Bitsy games take place in rooms made up of a grid of 16 by 16 tiled sprites. Sprites are made up of an 8x8 grid of “pixels,” or colored squares and can contain two frames of animation. Rooms are made up of four different types of sprites: tiles, sprites, items, and an avatar. Tiles are used for non-interactive decorative elements, sprites are used for interactive objects, items are things that can be picked up, and the avatar is a character that players can navigate by moving up, down, left and right, one tile at a time using the keyboard. Dialog boxes reveal short snippets of pixelated text when the avatar interacts with items and sprites. All of the elements in a given room are constrained to a color palette consisting of three colors: a background color, a tile color, and a sprite color. Most often, the core gameplay of a Bitsy game consists of the player moving a character around a world, exploring rooms and interacting with objects to meet characters, examine objects, and reveal pieces of a narrative.

2) The Interface. Bitsy uses panels as a basic organizational unit to structure its interface. Panels are arranged horizontally in the browser and can be turned on or off via a toolbar above. Panels can be reorganized via dragging. Each panel covers a different aspect of development within the tool, quickly familiarizing designers to the basic tasks available to them. When the app is launched, five of the 13 panels available to the user are turned on by default. These panels are: About Bitsy, which introduces the engine, provides links to tutorials, basic instructions, and release info; Room, whose 16 by 16 tiled grid serves as both a room editor and testing environment; Paint, which serves as an editor for creating avatars, tiles, sprites, and items; Colors, which allows designers to determine the color palette of each room; and Download, which facilitates both the current game and importing existing games as HTML files. Both Room and Paint panels use arrow buttons to allow users to quickly scroll through all of the rooms, items, sprites, and items created. Another important panel is the exit panel, which allows designers to define exit and entry points within rooms, allowing players to travel through them.

3) Design Philosophy. Bitsy attempts to reduce design friction in service of its intended purpose: enabling developers to quickly and easily create spaces to explore. This philosophy is demonstrated by the selection and prioritized arrangement of its default panels. The first two design-oriented panels that appear in the engine’s interface are its Room and Paint panels. These panels’ prominently displayed tiled editors encourage designers to focus their attention on creating assets. Traditionally, asset creation is one of the most time-consuming aspects of world creation. Bitsy addresses this a few ways. One way is through its constraints. By limiting designers to three colors, 8 by 8 sprites, and 16 by 16 rooms, and two frames of animation, Bitsy dramatically reduces the number of design decisions involved in the creative process. These restrictions also dramatically simplify the interface itself, for example, allowing an animation editor to fit directly below the sprite editor. By restricting all assets to be the same size and pixel dimensions, assets can be assured to fit within given bounds, which tightens the interface and establishes an instantly recognizable visual vocabulary.

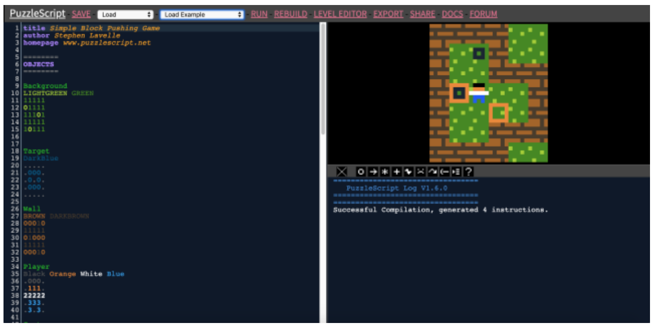
IV. PuzzleScript

A. History

Released in 2013 by independent game developer Stephen Lavelle, PuzzleScript is an open-source tile-based puzzle game engine. PuzzleScript was originally envisioned as a tool to create games similar to ZZT [24], an ASCII character action-adventure game with an active development community released for MS-DOS in 1991 by Tim Sweeny [4]. The engine was also heavily influenced by the syntax of ruleset defined in the game T in Y World, A Ludum Dare game by Tom Murphy in 2012 [26], which ultimately lead Lavelle to reimagine it as a tool for creating pure puzzle games. Although Lavelle states that PuzzlesSript is not a general purpose game engine, or even a general purpose puzzle game making tool [17], the vocabulary of possible interactions is quite expressive. Lavelle maintains a curated gallery, which currently consists of over 100 PuzzlesSript games that demonstrate the engine’s range, including quite a few made by notable independent game developers [16]. PuzzleScript has also been used in a number of published papers on automatic level generation [6].

B. The Engine

1) The Basics. PuzzleScript has been described as a tool for making “pushing stuff around” games [7]. This is most likely due to the fact that PuzzleScript’s declarative scripting language is well suited for creating games similar to Sokoban, a type of video game where the player pushes around crates to solve puzzles. In fact, the rules of the Sokoban-like puzzle game the engine uses to demonstrate PuzzleScript basics is expressed in just one line of PuzzleScript code ([> Player | Crate] -> [> Player | > Crate]). Although quite a few PuzzleScript games follow the conventions of Sokoban (turn-based, grid-based, moving a character, pushing and pulling objects) the engine supports



```
title Simple Block Pushing Game
author Stephen Lavelle
homepage www.puzzlescript.net

========
OBJECTS
========

Background
LIGHTGREEN GREEN
11111
01111
11101
11111
10111


Target
DarkBlue
.....
.000.
.0.0.
.000.
.....

Wall
BROWN DARKBROWN
00010
11111
01000
11111
00010

Player
Black Orange White Blue
.000.
.111.
22222
.333.
.3.3.
```

```
================================
   PuzzleScript Log V1.6.0
================================
================================
Successful Compilation, generated 4 instructions.
```

Fig. 3. A screenshot of PuzzleScript's interface, including its code editor, testing environment, and console.

many other types of interactive scenarios (e.g., moving multiple characters or controlling a cursor in a match three game). PuzzleScript games all use the same font and black and white title screen including the game's title, author, instructions, and options for starting a new game or continuing a previously started game, when appropriate. The instructions define the controls available to both designers and players: "arrow keys to move, x to action, z to undo, r to restart." The engine's visual vocabulary is similarly constrained: all objects within a PuzzleScript are made up of a grid of five by five "pixels" or blocks of color, giving PuzzleScript games a distinct look. The engine also has a simple generative tool for creating in-game sounds that add to PuzzleScript's retro aesthetic.

2) The Interface. PuzzleScript's editor is divided into four sections: a menubar at the top of the page, a code editor on the left, and testing environment and console on the right. Unlike Bitsy and Twine, which attempt to avoid requiring its developers to write code, PuzzleScript embraces programming. Everything in a PuzzlesScript game, from its assets and sounds to levels and rules, is defined using simple, concise code. PuzzlesScript code is divided into eight sections organized by default in the following order: Metadata, Objects, Legend, Sounds, Collision Layers, Rules, Win Conditions, and Levels. Metadata includes the game's title, author, and website, used by the game's title screen. In Objects, all game elements and their visual representation are defined by a name, list of its colors, and a 5 by 5 grid of numbers referencing its list of colors, defining its sprite. The Legend section defines single-character names for objects used the Levels section. The Sounds section allow developers to assign sounds to in-game events. In Collision Layers, developers can define lists of objects that collide with each other when moved. The Rules section is where developers define all the game logic for their game, referencing previously defined objects. Win Conditions also reference game objects to specify when a level has been completed. Finally, in the Levels section, the layout of each level is defined using a grid of characters referencing the single-character names defined in the Legend section.

3) Design Philosophy. In contrast to Twine and Bitsy, which does everything possible to shift its developers' attention away from programming and interactivity and towards creating content, PuzzleScript embraces code and interactivity by putting it front and center. This is made explicit on the first page of its documentation: Rules 101. Within the first paragraph of the first page, PuzzleScript has introduced a line of code and an animated GIF visualizing the resulting behavior, thus communicating a core tenet of its design philosophy: from simple rules can come complex systems. Developers quickly start to learn that code in PuzzleScript express rules and interactions between objects. Although PuzzleScript prioritizes programming and its syntax may initially seem foreign, it's also strikingly simple and presented in an accessible, easy to understand manner.

The style and tone of PuzzleScript's documentation communicate another principle of its design philosophy: to encourage experimentation. Rather than explaining the language by separating it into its discrete components and describing each, PuzzleScript invites developers to take a more playful approach. After demonstrating the one line of code that articulates Sokoban's ruleset, the documentation states, "Let's play with the above line of code - what would happen if we changed the direction of the arrows?" This is again followed by an animated GIF, showing how a pushing interaction changes to a pulling interaction. Only after this initial experimentation does the documentation explicitly explain how the engine works step-by-step.

V. Game Metadata

Included in this study is the metadata from 5,001 Twine, Bitsy, and PuzzleScript games gathered from itch.io in January 2019. Game metadata on itch.io is entered by the developer when publishing a game to the platform. The data collected consists of the following fields: Title, URL, Description, Price, Status, Game Platforms, Genres, Made with, Tags, Average session, Languages, Accessibility, Author, Author, URL, Author Site, and Author Twitter. Additional information, such as the number of games each developer published was also collected.

A. Twine Metadata

All 2,829 Twine games published to itch.io as of January 10th, 2019 that were identified as being made with Twine were used for this study. Of these games, only 50, or 1.7%, were paid, ranging from $0.10 to $599. 11 games were $1, the most common paid price for a Twine game on the platform. These 2,829 games were created by 1,940 developers, for an average of 4.4 games on the platform and an average of 27.0 followers. The top five tags used to describe Twine games aside from "Twine" were: "Text based" (727), "Story Rich" (250), "Short" (236), "Horror" (227), and "Fantasy" (141). Other tags of note were "LGBT" (117) at #9, "Female Protagonist" at #11, "Dating Sim" (34) at 34, and "Transgender" (31) at 34. The most prolific Twine developer on the site was Naomi Norbez with 20 Twine games published. Ten other developers had 10 or more published Twine games.

B. Bitsy Metadata

The 1,664 Bitsy games analyzed in this study were created by 923 unique developers for an average of 5.6 total games posted to itch.io. Bitsy developers averaged 35.8 followers. Of the Bitsy games published, all but one was free, which cost $1. The top five tags used to describe Bitsy games were "Pixel Art" (402), "2D" (267), "Short" (234), "8-Bit" (170), and "1-bit" (138). Other tags of note were "cats" (68) at #10, "Atmospheric" (63) at #11, and "Cute" (57) at #14. The most prolific developers were "onion," with 32 Bitsy games published and "Sean," with 30.

C. *PuzzleScript Metadata*

The 508 PuzzleScript games analyzed in this study were created by 308 developers. On average, a developer who published a PuzzleScript game also published 6.8 total games on itch.io and averaged 28.3 followers. PuzzleScript games ranged in price from free to $9, but like other TOGE games, paid games were quite rare, making up only 1.9% of the PuzzleScript games released on the platform. The top five tags used to describe PuzzleScript games aside from “PuzzleScript” were: “Pixel Art” (133), “blocks” (106), “Short” (104), “Singleplayer” (98), and 'sokoban' (93). Other tags of note were “Difficult” (92) at #7, “mind-bending” (11) at 20, and “Abstract” (9) at 22. Rosden Shadow was the most prolific PuzzleScript developer on the site with 84 published PuzzleScript games! Second was st33d with 12.

TABLE I. TOP 25 TAGS FOR GAMES ON ITCH.IO

Twine	***Bitsy***	***PuzzleScript***
Twine: 1698	Bitsy: 1237	PuzzleScript: 430
Text based: 727	Pixel Art: 402	Pixel Art: 133
Story Rich: 250	2D: 267	blocks: 106
Short: 236	Short: 234	Short: 104
Horror: 227	8-Bit: 170	Singleplayer: 98
Fantasy: 141	1-bit: 138	sokoban: 93
Multiple Endings: 134	Minimalist: 100	Difficult: 92
Sci-fi: 119	Walking simulator: 89	Top-Down: 87
LGBT: 117	Exploration: 83	16-bit: 77
Experimental: 105	cats: 68	2D: 64
Female Protagonist: 96	Atmospheric: 63	8-Bit: 39
Comedy: 95	Story Rich: 63	Puzzle-Platformer: 29
Romance: 94	Narrative: 57	Casual: 19
Narrative: 91	Cute: 57	maze: 19
artgame: 89	Singleplayer: 54	Turn-based: 18
Singleplayer: 89	Experimental: 51	Minimalist: 16
Atmospheric: 84	Abstract: 49	Retro: 13
meaningful-choices: 84	Retro: 47	Action-Adventure: 12
weird: 77	Casual: 46	3D: 11
Funny: 72	Horror: 44	mind-bending: 11
Dark: 70	bitsyjam: 43	Arcade: 11
Mouse only: 65	Lo-fi: 43	Abstract: 9
Mystery: 63	artgame: 37	indie: 8
Abstract: 60	Relaxing: 34	Horror: 8
Minimalist: 60	Non violent: 34	Colorful: 8

Fig. 4 The top 25 tags used to describe games made with TOGEs on itch.io. Each tag is followed by its frequency.

D. *Discussion*

A few trends can be seen from a comparative analysis of this data.

- Although the first few most-used tags for each engine were similar or predictable, interesting differences can be found the further down the list, which hint at each engine’s personality (e.g., Bitsy’s “cats” and “cute”).
- The popularity of tags “LGBT,” “Female Protagonist,” and “Transgender,” found to describe Twine games supports Friedhoff’s assertion that the engine has become a hotbed for exploring issues of marginalization, queernesss, and discrimination [11].
- PuzzleScript developers, on average, published more games than Twine or Bitsy developers. Whether this can be attributed to the ease of PuzzleScript to make complete games (since they tend to be less content heavy), or the engine attracts a certain type of developer is unclear.
- Bitsy developers had significantly more followers on itch.io than PuzzleScript or Twine developers. One possible reason for this may be that Bitsy developers have more invested in the itch.io platform, since Bitsy itself was launched on the site and uses many of its community building features.
- TOGE developers aren’t in it for the money. Very few developers charge for their games.

VI. DEVELOPER SURVEYS

An online survey was given to TOGE developers. Participation was solicited by contacting developers in one of four ways: contact via email through addresses found on developer websites provided in the “Author Site” field in the game metadata, via Twitter from the “Author Twitter” field in the game metadata, posting on tool-specific developer forums, and posting on tool-specific developer Discord servers. This outreach resulted in 163 TOGE developers completing the survey.

A. *Twine Survey Results*

The 61 Twine developers that completed the survey were a mean age of 28, ranging from 20 to 59. 45.9% identified as male, 32.8% as female, and 21.3% identified as non-binary. Respondents came from 13 countries. 37.8% lived in the USA, 24% in the UK, 8.1% in Spain, and 8.1% in Canada. When asked about their experience with game development, 49.2% stated that they had been developing games for 1-3 years (second was 3-5 years with 21.3%), 39.3% had released 1-3 games before (5-10 was second with 26.2%), and 37.7% said they had no experience programming. 41.0% identified as hobbyists (amateur was second with 29.5%) and when asked if they had any experience with other game engines, 50.8% said Unity, and Construct, Bitsy, and GameMaker tied for second at 18.0% each. When creating games, 73.8% said they work alone and 37.7% said it takes them 12-48 hours to complete a game (second most popular was 48-120 hours with 26.2%). When asked about what aspects of game creation are most interesting to them, 85.2% said creating narratives, and 59.0% said mechanics and systems. When asked what inspires them to make games, 70.5% said creating fictional worlds and 63.9% said personal experience. 67.2% of the Twine developers said they want their players to be emotionally affected and 60.7% wanted them to be entertained. Finally, when asked why they make games, 90.2% as a form of expression and 77% said to explore interactivity.

B. *Bitsy Survey Results*

The 59 Bitsy developers that completed the survey were a mean age of 26.7, ranging from 14 to 48. 59.3% identified as male, 15.3% as female, and 25.4% identified as non-binary. The respondents came from 18 countries with 35.6% from the USA, 16.8% from the UK, and 10.2% from Canada. When asked about their experience with game development, 32.3% stated that they had been developing games for 1-3 years (second was <1 year with 18.6%), 35.6% had released 1-3 games before (10+ was second with 33.9%), and 27.1% said they had no experience programming. 62.7% identified as hobbyists (amateur was second with 37.3%) and when asked if they had any experience with other game engines, 55.9% said Unity, 49.2% said Twine, and 37.3% said GameMaker. When creating games, 83.1% said they work

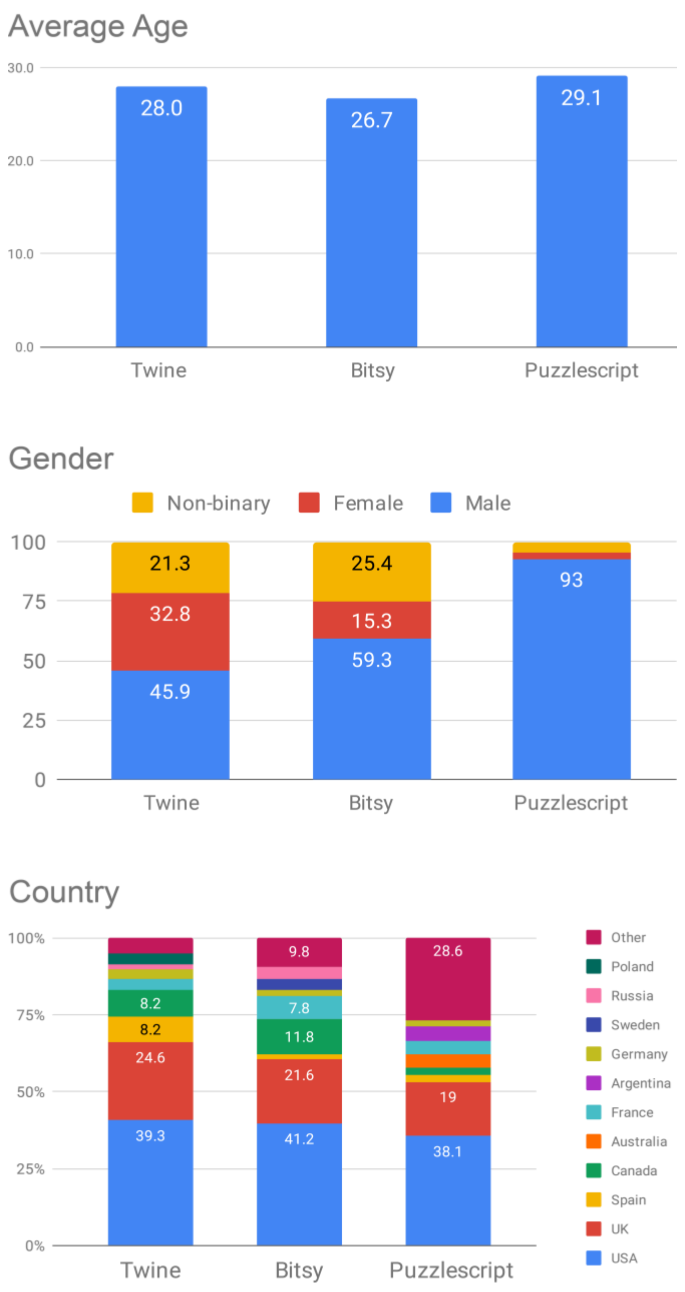


Fig. 5 Average age, gender and country statistics of Twine, Bitsy, and PuzzleScript developers on itch.io.

alone and 35.6% said it takes them 6-12 hours to complete a game. When asked about what aspects of game creation are most interesting to them, 69.5% said creating narratives, 67.8% said mechanics and systems, and 67.8% also said aesthetics. When asked what inspires them to make games, 59.3% said creating fictional worlds and 57.6% said personal experience. 62.7% of the Twine developers said they want their players to be emotionally affected and 59.3% wanted them to experience something new. Finally, when asked why they make games, 89.8% as a form of expression and 54.2% said to explore interactivity.

C. *PuzzleScript Survey Results*

The 43 PuzzleScript developers that completed the survey were a mean age of 29.0, ranging from 14 to 48. 93.0% identified as male, 2.3% as female, 4.7% identified as non-binary, and 5% identified as agender. Respondents lived in 17 counties, the most being 37.2% in the USA and 18.6% in the UK. When asked about their experience with game development, 34.9% stated that they had been developing games for 3-5 years (second was 1-3 years with 25.6%), 34.9% had released 1-3 games before (10+ was second with 32.6%), and only 4.7% said they had no experience programming (the highest answer being 10+ years). 72.1% identified as hobbyists (amateur was second with 44.2%) and when asked if they had any experience with other game engines, 53.5% said Unity, 37.2% said GameMaker, and 37.2% said Twine. When creating games, 83.7% said they work alone and 41.9% said it takes them 12-48 hours to complete a game (second most popular was <6 hours with 23.3%). When asked about what aspects of game creation are most interesting to them, 88.4% said creating mechanics and systems, and 48.8% said aesthetics. When asked what inspires them to make games, 67.3% said creating new mechanics and systems and 34.9% said personal experience. 62.8% of the Twine developers said they want their players to be entertained and 62.8% wanted them to experience something new. Finally, when asked why they make games, 74.4% as a form of expression and 51.2% said to explore interactivity.

D. *Discussion*

The following observations were made from the collected responses.

- Overall, Bitsy developers were the youngest and least experienced, however, on average, they released more games than Twine developers and created them the fastest.
- A significant percentage of both Twine (21.3%) and Bitsy (25.5%) developers identified as non-binary. Twine had the highest percentage of female developers (32.8%).
- Twine games were reported as the most time consuming TOGE to make games with, even though there is little to no programming involved.
- PuzzleScript developers were the oldest and overwhelmingly male. They also had significantly more programming experience than the other two groups. They were most likely to say creating a game takes less than six hours to complete.
- A number of answers were consistent across groups. Most developers were from USA and the UK. Unity, GameMaker, and Twine seemed to be the most commonly used engines amongst all groups.
- In general, developer interests in game development matched the TOGE design philosophies identified. PuzzleScript developers were most interested in creating new mechanics and systems, Bitsy developers were most likely to be concerned with aesthetics, and Twine developers were most likely to be interested in creating stories.

VII. CONCLUSION

This paper identifies a trend in game development tools that is moving at odds with the game industry at large. Developers of TOGEs are choosing simplicity, elegance and artful expression over complexity and the race for evermore polygons. An active, passionate, diverse group of TOGE developers have proven that embracing constraints and celebrating small experiences can be a successful strategy for creating tools that are immediately accessible, stimulate creativity, and reach a wide audience. By highlighting the work of these tool makers, the design philosophies they've made concrete, the communities they've helped form, and the experiences they've enabled, maybe we can free future tool makers from thinking they have to create an engine that can do it all. And perhaps we can encourage the community to appreciate these tools as artful forms of expression in and of themselves.

ACKNOWLEDGMENT

The author would like to thank all of the survey participants who for the time and data they generously provided.

REFERENCES

[1] A. Dixon, "How small game makers found their community with Bitsy," 2018. [Online]. Available: https://www.rockpapershotgun.com/2018/02/23/how-small-game-makers-found-their-community-with-bitsy/ [January 11th, 2019].

[2] A. Le Doux, "Bitsy," 2018. [Online]. Available: http://ledoux.io/bitsy/editor.html [January 11th, 2019] .

[3] A. Khalifa, Magda Fayek. "Automatic puzzle level generation: A general approach using a description language." In Computational Creativity and Games Workshop. 2015.

[4] A. Anthropy. *Rise of the videogame zinesters*. Seven Stories Press. 2012.

[5] A. Anthropy. *ZZT*. Boss Fight Books. 2014.

[6] C. Lim, D. Fox Harrell. "An approach to general videogame evaluation and automatic generation using a description language." Computational Intelligence and Games (CIG), 2014 IEEE Conference on. IEEE. 1-8. 2014.

[7] G. Smith, "PuzzleScript Is A Simple Language For Making Puzzle Games," 2013. [Online]. Available: https://www.rockpapershotgun.com/2013/10/14/puzzlescript-is-a-simple-language-for-making-puzzle-games/ [October 2013].

[8] H. Epstein, "Drawing in the 21st Century," 2015. [Online]. Available: https://killscreen.com/articles/drawing-21st-century/ [January 11th, 2019].

[9] I. Maclarty, "Vertex Meadow," 2015. [Online]. Available: http://www.vertexmeadow.xyz/ [January 11th, 2019].

[10] itch.io, "Download the latest indie games," 2019. [Online]. Available: https://itch.io/ [January 9th, 2019].

[11] J. Friedhoff. Untangling Twine: A Platform Study. In DiGRA conference. 2013.

[12] K. Imtiaz, "God of War PS4 Development Team Is 3x God of War 2, Has 270 Devs Working On It," 2017. [Online]. Available: https://gearnuke.com/god-war-ps4-development-team-3x-god-war-2-270-devs-working/ [January 10th, 2019].

[13] L. Alexander, "Game creation for the masses: What's next for Twine," 2014. [Online]. Available: https://www.gamasutra.com/view/news/227313/Game_creation_for_the_masses_Whats_next_for_Twine.php [January 9th, 2019].

[14] M. B. Sauter, "Popular video games in 2018: A list of the 25 best-selling titles this year," 2018. https://www.usatoday.com/story/tech/gaming/2018/12/13/popular-video-games-2018-25-best-selling-titles-year/38672903/ [January 10th, 2019].

[15] N. Carroll, "Interview with the Creator of Twine: Chris Klimas," 2014. [Online]. Available: https://nilsoncarroll.wordpress.com/2014/03/18/interview-with-the-creator-of-twine-chris-klimas/ [January 8th, 2019].

[16] P. Davison, "Make Your Own Puzzle Games with PuzzleScript," 2013. [Online]. Available: https://www.usgamer.net/articles/make-your-own-puzzle-games-with-puzzle-script [January 11th, 2019].

[17] PuzzleScript, "About," 2018. [Online]. Available: https://www.puzzlescript.net/Documentation/about.html [January 11th, 2019].

[18] R. Weber, "On Reflections: First interview with the Ubisoft studio's new MD," 2013. [Online]. Available: https://www.gamesindustry.biz/articles/2014-02-26-on-reflections-first-interview-with-the-ubisoft-studios-new-md [January 10th, 2019].

[19] R. Koster, "The cost of games," 2018. [Online]. Available: https://venturebeat.com/2018/01/23/the-cost-of-games/ [January 10th, 2019].

[20] R. Mehta, Meng Zhu. "Creating when you have less: The impact of resource scarcity on product use creativity." Journal of Consumer Research, 42(5), 767-782./ 2015.

[21] S. Lavelle. "flickgame," 2015. [Online]. Available: https://www.flickgame.org/ [January 11th, 2019].

[22] S. Lavelle, "plingpling," 2015. [Online]. Available: https://www.plingpling.org/ [January 11th, 2019].

[23] S. Lavelle, "tinychoice," 2015. [Online]. Available: https://www.plingpling.org/ [January 11th, 2019].

[24] S. Lavelle. Personal interview. 2018.

[25] T. Chapman, "God of War Sold More Copies in 3 Days Than Horizon Did in 2 Weeks," 2018. [Online]. Available: https://screenrant.com/god-war-ps4-fastest-selling-exclusive/ [January 10th, 2019].

[26] T. Murphy, "T in Y World," 2012. [Online}. Available: http://tinyworld.spacebar.org [January 11th, 2019].

[27] Twine Wiki, "What Can You Build With Twine?" 2017. [Online]. Available: https://twinery.org/wiki/twine2:what_can_you_build_with_twine [January 11th, 2019].

[28] "Unity Game Dev Courses: Swords and Shovels." [Online]. Available: https://unity3d.com/swordsandshovels#tbp-coursespagetile [January 10th, 2019].

[29] Unreal Engine Forums, "PDF for UE4 Documentation Available?" 2018. [Online]. Available: https://forums.unrealengine.com/community/general-discussion/6602-pdf-for-ue4-documentation-available/page4?7348-PDF-for-UE4-Documentation-Available=&viewfull=1 [January 10th, 2019].

[30] Wikipedia, "Twine (software)," 2018. [Online]. Available: https://en.wikipedia.org/wiki/Twine_(software) [[January 11th, 2019].