Villanelle: An Authoring Tool for Autonomous Characters in Interactive Fiction

Chris Martens and Owais Iqbal

martens@csc.ncsu.edu omiqbal@ncsu.edu
North Carolina State University, Raleigh, NC USA
https://go.ncsu.edu/poem

Abstract. Our goal is to discover tool and language design principles that enable powerful, usable autonomous character authorship for diverse audiences of storytellers. This paper describes the Villanelle project, an approach to interactive narrative authoring that supports seamless integration of autonomous characters into choice-based storytelling. We present our computational model based on behavior trees uniformly for scripting agent interaction, user interaction, and narrative events; our stand-alone authoring tool, which provides an integrated development and testing environment for authoring with this model; and our JavaScript API for web-based development, demonstrating the expressiveness and simplicity of our approach through two case studies.

1 Introduction

The rise in popularity of interactive narratives has led to the introduction of authoring tools that aim to bridge the gap between two different skill-sets required for creating an interactive narrative: narrative design (for authoring the narrative, world and characters) and programming (for realizing the narrative and the different mechanisms the author has in mind). Tools like Twine [10] have gained wide user bases among underrepresented storytellers and game makers due to their usability without programming experience [5]. These tools allow an author to quickly write and test the narrative ideas that they have in mind without focusing the majority of their attention on implementation details.

Meanwhile, there is active and growing interest in creating *procedural* play systems that promote player interest through worlds that continue to change and grow without player intervention, yet respond to player input [24]. One way to achieve this effect is through *autonomous NPCs* (non-player characters) who act according to their own plans and goals and create emergent interactions among themselves. The intelligent narrative research community has made significant advances in storytelling with autonomous NPCs, including reactive systems such as ABL [11] underlying the landmark interactive drama Facade [12], planning-based systems that regenerate narrative arcs based on player decisions [1, 20, 19], and the social practice systems encoded in CiF and Versu [14, 4].

However, there is a significant gap between the potential expressiveness afforded by autonomous characters in interactive narrative and the availability

of authoring tools that make these techniques approachable and usable in the same way that Twine and Inform have made hypertext and parser-based storytelling authorable. Current tools lie at different ends of the approachability-expressiveness scale when it comes to intelligent character authoring, which is to say that if a tool has the capability to encode a complex behavior for an autonomous character, then it generally also has a steep learning curve.

This paper presents *Villanelle*, a framework and tool for authoring NPC behavior in interactive narrative works, which aims to bridge this gap. Villanelle adopts *behavior trees* (BTs) as a computational foundation for authoring character behaviors and player interactions. Behavior trees have shown effective for scripting AI characters and proven by wide adoption in AAA games [16] and game creation frameworks like Unity 3D and Unreal Engine. Behavior tree proponents cite how easy they are to create, maintain, and scale, allowing designers to quickly be able to create the behavior they want in autonomous characters without getting lost in minute implementation details. Designers can craft reusable subtrees of behavior to be used for different characters or repeated multiple times in the same tree. Behavior trees allow authors to focus on the overall agent behavior they want to achieve.

Villanelle uses behavior trees not only for scripting NPCs in interactive narrative, but also for writing choice frames, game rules, and the outcomes of player actions in the interactive narrative. This choice demonstrates our philosophy of *language minimalism*, presenting a minimal basis of programming constructs that, once learned, can be composed to fulfill a wide range of authoring needs. We hypothesize that once an author grasps the basics of implementing a behavior tree, they will be able to author new, experimental narrative experiences that rely on believable and responsive virtual characters. In the long run, a successful execution of this concept will offer a healthy balance of expressiveness and approachability.

Our key contributions in this paper are as follows: (a) a reproducible description of how BTs can be used to author story characters and branching choice structures in the context of text-based interactive fiction (Section 3); (b) a system description of the Villanelle authoring tool (Section 4) and application programming interface (API) (Section 5); (c) a demonstration of Villanelle's capabilities through two case studies, one using the stand-alone authoring tool (Section 4) and one using the underlying API (Section 5).

2 Related Work

There is an extensive body of research on authoring tools for developing interactive narrative, across a wide variety of goals for the resulting narrative works. Grow et al. [6] compare three tools specifically for authoring interactive virtual agents: Bryson et al.'s BOD/POSH [2], Dias et al.'s FAtiMA [3], and Mateas and Stern's ABL [11]. These authoring tools were evaluated on an example referred to as the "Lost Interpreter" scenario in which the player, as an armed soldier in occupied territory, must show a photograph to locals in order to find

their lost interpreter. However, none of these tools were evaluated in terms of their ability to express *autonomous* behavior, i.e. actions taken by NPCs that are *not* in direct reaction to player actions. Villanelle's adoption of BTs targets this mode of use in particular, and suggests the need for a wider range of case study scenarios with which to evaluate interactive narrative technology.

A number of other tools for interactive narrative authoring have been developed and described in academic literature, such as Scribe [15], IDTension [26], Narratoria [27], and Mimmisbrunnur [25]. These tools place varying levels of emphasis on NPC autonomy. Among these, Versu [4], CiF [14] (and its successor Ensemble [22]), and the Spirit AI Character Engine demoed at AIIDE 2018 [21]) are probably the closest in their goals to Villanelle; however, all of these tools have more of a focus on imparting characters with believable emotional and social intelligence. In contrast, Villanelle is agnostic to the particular set of actions that characters can carry out (whether they be related to mood changes, logistics like moving between locations and manipulating items, or insulting or befriending other characters) and is more concerned with the mechanics of authoring; i.e. on evaluating BTs as an computational model for coordinating NPC behaviors.

Behavior trees have seen widespread adoption in the mainstream gaming industry, particularly for NPC AI in real-time strategy and first-person shooter games [8], and efforts have been made to make them easy for designers to author through tools like BehaviorShop [7] and Unity3D's Behavior Designer. In the IN context more specifically, Kapadia et al. conducted an evaluation of behavior trees for narrative authoring and user interaction [9], comparing them to a story graph approach. This study was done using the Unity3D engine and an existing story framework created by the authors. However, their user study found that expert programmers still took multiple hours to develop a relatively minimal example. Our approach to handling user interaction with BTs requires less authoring overhead, and we anticipate that a similar example would take much less time to author.

3 Villanelle's Behavior Trees

The Villanelle project takes a "language-based" approach to authoring, which means we distinguish between the computational model afforded to authors and its implementation as an authoring tool, which included syntactic sugar and integrated editing environment support. Villanelle's computational model uses behavior trees (BTs) to represent branching narrative structures as well as NPC AI. We chose BTs based on their wide adoption in the games industry by designers [8], who vouch for their ease of development as well as reusability for encoding different characters with the same behavior. To minimize the learning curve, Villanelle chooses to implement only the minimal basic constructs of behavior trees: sequencing, selection, conditions, and actions. The implementation is also done in a functional programming manner, using the formalism described in previous work (citation omitted for blind review). We recapitulate this formalism in this section.

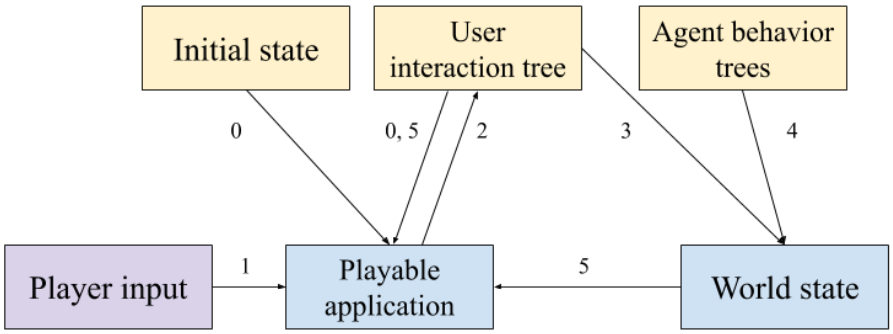

Fig. 1. A diagram of the game loop architecture in Villanelle. Yellow nodes are authored constructs, and blue nodes are run-time artifacts. The two edges labeled 0 represent configuring the initial story world for the player using the authored user interaction tree and initial state. Edge 1 indicates the player making a choice. Edge 2 propagates this choice to the user interaction tree, which updates the world through edge 3. Edge 4 represents BTs for each agent collectively taking their "turns" and modifying the world state. Finally, the to edges labeled 5 indicate rendering the updated world state to the player, potentially offering different choices based on conditions in the user interaction tree.

Villanelle was designed for interactive fiction, which traditionally works in a turn-based manner. Therefore, the actions and subsequent changes to the state of the world occur over discrete time steps. Upon selection of an action to perform by the player, Villanelle executes the behavior tree mapped to that action as well as the next step of each behavior for each agent in the game (see Figure 1).

Villanelle uses behavior trees (BTs) as its underlying computational model. We next describe the specific BT constructs that make up this computational model using the example in Figure 2. The types of nodes that Villanelle uses are primitive *actions*, which appear at the leaves of the tree, *sequence* and *selector* nodes consisting of two or more children, and *guards*, which have a single child node. Every node type is implemented as a function that returns a Status of `SUCCESS`, `FAILURE` or a `RUNNING` upon execution, which sometimes depends on the status returned by its children.

3.1 Primitive Actions

Action nodes are responsible for mutating the world state. Actions need to specify their preconditions and effects. A precondition is a function that will inspect certain variables in the world state and return a boolean value. If it is true, the effects parameter gets executed and if it is false, the Tick returns a status of `FAILURE`. Effects are responsible for all observable changes, including printing text that the player will see or changing variables that other agents may react to.

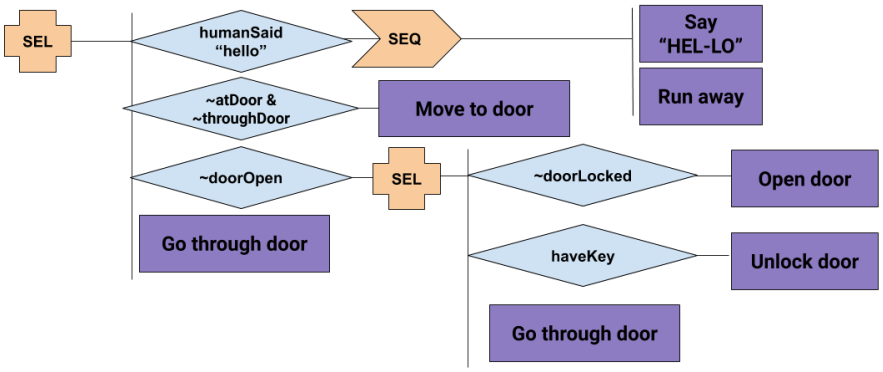

Fig. 2. An example behavior tree for an agent. Composite nodes are color-coded orange and shaped as plus signs or arrows, condition guards are colored blue and diamond-shaped, and primitive actions are purple rectangles.

As an example, the following code specifies an action node with a precondition that checks whether a door is unlocked, and if it is, opens the door. This code implements the condition and action seen in the middle-right of Figure 2.

```
condition: not doorLocked
effects: doorOpen := true
```

3.2 Composite and Guard Nodes

A 'Composite' is any node with two or more children. The two types of composites currently implemented in Villanelle are *sequences* and *selectors*. A sequence node executes its children sequentially until one of them returns a FAILURE. This node returns a SUCCESS on successful execution of all children. If a child node is in the RUNNING status, this node will return a RUNNING status as well. A selector node is the inverse of the sequence: it executes children sequentially until one of them returns a SUCCESS and hence the name (it 'selects' a successful node from its children). The node fails if it doesn't find a single successful node. The case for when a child returns RUNNING is the same as for the sequence node.

Finally, Villanelle also provides a Guard node, which allows the author to couple a composite node with a precondition. If the precondition fails, this node would fail else it would return whichever status the Composite node returns.

The following example represents a subtree of Figure 2 and uses conditions, sequences, and selectors in combination.

```
selector:
    condition: humanSaid "hello"
        sequence:
```

```
        - print "'HEL-LO,' says the robot."
        - atDoor := false
  condition: not atDoor and not throughDoor
    - atDoor := true
```

3.3 Agents

An agent is a structural entity that consists of a behavior tree and variables specific to the agent. Variables are still written to the blackboard, but they are scoped to the agent. Agents provide an easy to understand way to label behavior trees, as the typical use case would be to attach a different behavior tree for each major character in the narrative. It is not limited to only characters though, as the author could also provide a behavior tree for major narrative events in the game with a "Director" agent.

3.4 Player Interaction with the Agents and the World

Villanelle supports the use of BTs for specifying player interactions predicated on the state of the world. The author does this by defining *user interaction trees* that the framework runs after all the agent trees have run. See Figure 3 for an example. There are two authorable components of player interaction: what the player sees, and the set of choices available to the player (coupled with their effects). *What the player sees* may contain a description of the current scene and the current state of some of the game's variables. *Player choices* consist of a list of actions the player can perform given the current state, as well as the effects of each choice and the text description of the action having been carried out.

4 Standalone Authoring Tool

In prior work on usable authoring tools [15], researchers advocate for "one centralized tool in which [all] authoring functions take place." Accordingly, we developed a standalone cross-platform desktop tool for writing and debugging interactive narrative works. This tool includes live visualization of all behavior trees created by the author and live rendering of the game. Our goal is to allow authors to quickly prototype their ideas with the built-in editor and play the game immediately after making their changes without requiring additional steps. With live visualization of the trees, the authors can graphically understand the structures that they are building and use the live error reporting to help fix syntax and semantic issues instantly. If compilation succeeds, the author can play their game in the tool and see the statuses of the different nodes of the behavior trees as the game progresses. We implemented the tool using the Electron framework for creating a cross-platform desktop application, the JavaScript React framework to handle rendering the application, and Palantir's Blueprintjs for the user interface.

```
User Interaction:
   - condition: botAtDoor
     sequence:
       - description: "There is a little robot here."
       - user action:
           action text: "Say hello"
           effect tree:
               effects:
                   - sayHello := true
   - user action:
       action text: "Wait"
       effect tree:
           effects:
               - none := true
```

Fig. 3. This example presents the user with two possible choices, "Say hello" and "Wait", where the former is only active when the Robot agent is at the door. The effects of saying hello set a variable that the Robot can respond to in its next turn.

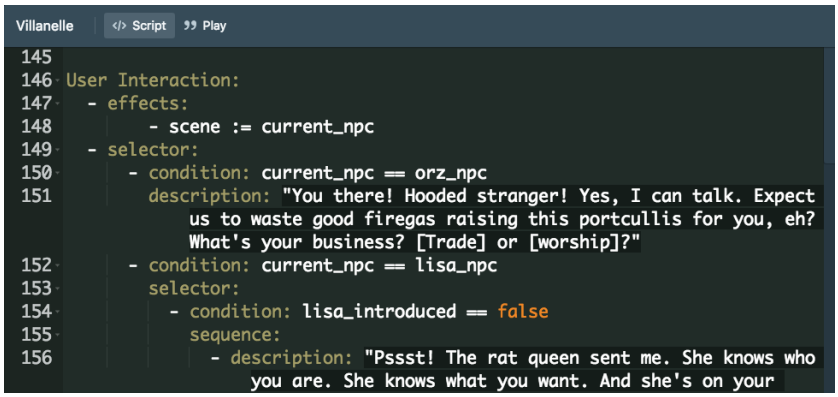

Fig. 4. A screenshot of the Editor tab.

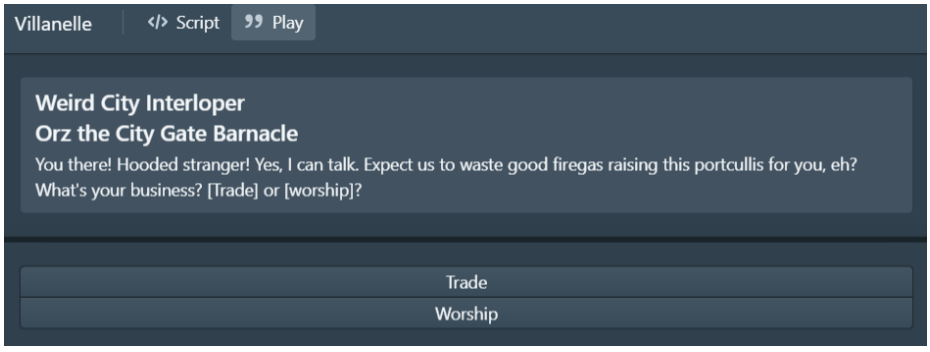

Fig. 5. A screenshot of the Play tab corresponding to the code in Figure 4.

4.1 Case Study: Weird City Interloper (Port)

Every new feature in the standalone tool was tested by developing a playable experience that uses it. We have created and tested several playable experiences with the tool, two of which we highlight here to demonstrate practicality and breadth. We will explain the features of the editor using a case study based on *Weird City Interloper*, a text adventure game by C.E.J. Pacian released in 2014. On the Interactive Fiction Database, it has 32 ratings, averaging 4.5 star reviews [18]. We chose to port this game to Villanelle to evaluate its usability for developing a choice-based exploratory game centering around conversation with NPCs. Each NPC's dialogue interface is controlled by a separate behavior tree; see Figure 6 for an example.

The Villanelle editor has two tabs, Script and Play (see Figures 4 and 5 for screenshots). On every change to the Script input, the game is rendered immediately in the Play tab. If there are any errors in the input, the compilation fails and an error message is displayed instead. The rendered game has two components: the text display and the player choices. The text display consists of the title of the game and scene, the scene description as given by the user interaction tree, and the effect text provided for any agent actions that run, if any. In the choice input pane, we render each choice authored in the user interaction tree as a button that will execute the associated behavior tree when clicked.

4.2 Editing Support

The script tab primarily consists of the editor, seen in Figure 4. The editor was realized using an open source embeddable code editor called Ace Editor. We used the built-in language mode for YAML, the syntax upon which we developed the Villanelle surface syntax. The Ace Editor also provides general features like a powerful search/replace functionality (which has regular expression support), highlighting other same tokens when one is highlighted and line numbers.

In a side panel next to the editor, a tree is rendered live with every change the user makes to the YAML in the editor. This tree is also responsible for highlighting the errors in the code structure if there are any. Every individual node which has children is expandable and collapsible. The following is how the different components are shown graphically (see Figure 6).

4.3 Debugging Support

We use *json-schema* and *ajv* libraries to perform error checks. We also run the condition and assignment expressions against the ANTLR4 grammar. Any errors in these checks are reported in a bottom bar with a dot indicating failure. If the error checks succeed, the bar turns green and includes a checkmark. Every change to the YAML input in the editor causes the error checks to be run again.

If the input is an invalid YAML schema, i.e. it violates any of the general YAML rules, the tree isn't rendered and a message is shown.

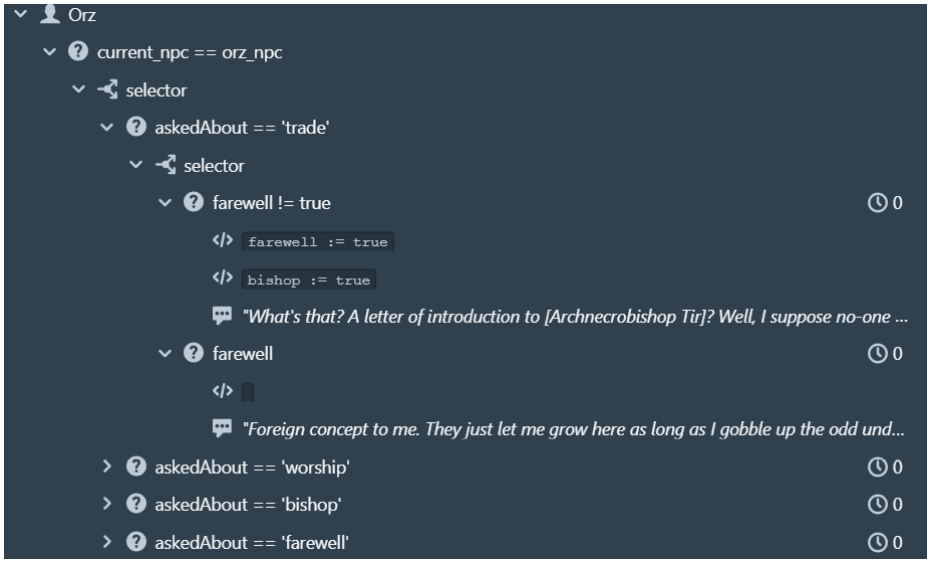

Fig. 6. The behavior tree of an agent NPC, visualized.

Tree Visualization Behavior trees for agent nodes are rendered under each agent. The visual structure of the behavior trees matches node for node the structure in the YAML input. However, the conditions in actions, sequence or selectors are represented as individual nodes themselves, with the associated behavior tree node rendered as a child. This was done to visually create a sense of 'gate-keeping' the conditions provide in terms of their coupling with nodes of a behavior tree.

The number of ticks an action node takes is displayed as a clock symbol on the right hand side of the corresponding condition node (if the action has no explicit condition node, a 'true' condition node is rendered).

Nodes which have errors with types or with the Villanelle YAML schema are reported as red nodes and their children are not rendered. The error message is displayed on hovering over the erroneous node.

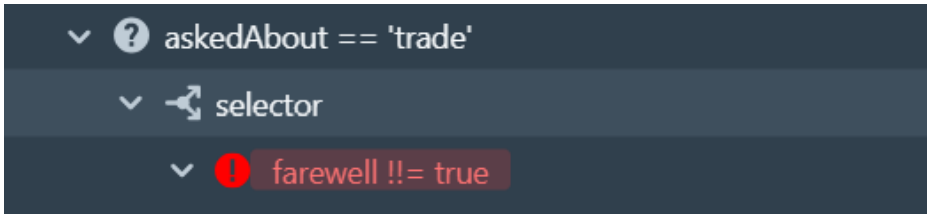

Fig. 7. The condition expression is syntactically incorrect

All ancestors of the erroneous node are automatically expanded so the author does not have to search on their own.

Tree Execution Visualization We support debugging by rendering the live execution of behavior trees during gameplay. As the author plays through the game, the tree nodes are highlighted based on how they were processed: green means SUCCESS, red means FAILURE, and orange means RUNNING. Every time the user takes an action, the statuses of the nodes change and the tree is refreshed showing the changes, giving the author live feedback.

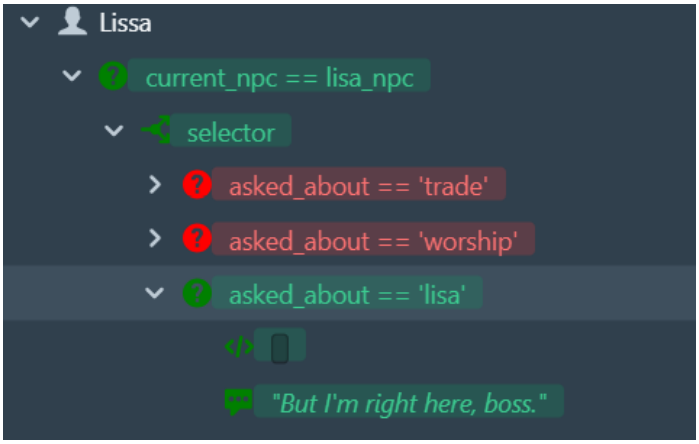

Fig. 8. The different statuses for the nodes show up as you play the game

5 Application Programming Interface

To support development of web-based interactive narrative experiences, we released an open-source web application programming interface (API) for Villanelle. This API gives authors the ability to create BTs, register initial world states, and execute the world engine, by calling JavaScript functions. The main game loop is called `worldTick()`, whose code is shown in Figure 9.

While the stand-alone editor provides minimal language features, such as variables that can hold strings, numbers, and boolean values, the API is more flexible and intended for advanced users. It supports modularity, behaviors that take parameters (as in [23]), and arbitrary data structures supported by TypeScript (e.g. arrays and dictionaries).

5.1 Case Study: Rime Royale (Original Game)

We showcase the expressiveness of Villanelle's API through *Rime Royale*, an original, browser-playable game developed by our lab. Rime Royale implements

```
export function worldTick() {
    // Execute each agent's behavior tree
    for (var i = 0; i < agents.length; i++) {
        var tree = agentTrees[agents[i]];
        if (!isUndefined(tree)) {
            setVariable("executingAgent", agents[i]);
            execute(tree);
        }
    }
    // Execute the user interaction tree
    runUserInteractionTrees();
}
```

Fig. 9. Code for a function provided by the API to execute all behavior trees defined by the author for agents and user interaction.

Fig. 10. A Screenshot of *Rime Royale*

a guildmaster roleplaying mechanic in which the player must assign NPCs with various strengths to missions that can succeed or fail; see a screenshot in Figure 10). Villanelle is used to implement the behavior of NPCs not assigned to missions, which act autonomously according to their personalities and preferences while other characters attempt the missions. Rime Royale was accepted for presentation in the AIIDE 2019 Playable Experience track (citation omitted for anonymous review), which shows evidence of the strength of its gameplay.

Rime Royale was developed over the course of one Spring semester by two undergraduate students, one responsible for art and narrative direction and one responsible for AI and gameplay programming. Their success provides evidence of Villanelle's support for innovative forms of gameplay.

6 Conclusion and Future Work

In this paper we presented Villanelle, an API framework and a standalone tool to use behavior trees not only to author character behaviors but every other part of an interactive narrative experience as well. Although we have not yet conducted a formal user evaluation, we have found through internal testing that Villanelle enables painless development of a wide range of reusable behaviors for autonomous characters. In future work, we would like to investigate scalability to large groups of NPCs in a social simulation akin to Prom Week [13] and formally compare Villanelle to other authoring tools using benchmarks in previous tool evaluations [9, 6].

Initial feedback from are target developer audience suggests that a number of additional features would be useful, some of which (like behavior parameterization and composite data structures) exist in the API but not the standalone tool. Other more foundational features include an extension to the behavior tree language that supports a stronger notion of reactivity through continuously monitored nodes (as in Unity's Behavior Designer implementation). We also plan to investigate the feasibility and utility of behavior generation through planning: BTs lend themselves very well to Hierarchical Task Network planning [17]. We could use this technique to automatically compose trees using from the available trees created by the author. Finally, in accordance with the evidence that debugging and reasoning are key usability principles, we plan to investigate reasoning principles for multi-agent systems authored with behavior trees. This includes debugging support for stepping, jumping, and rewinding (to analyze unexpected NPC interactions) as well as behavior model checking to validate (un)reachability of story states.

References

1. Aylett, R.: Narrative in Virtual Environments - Towards Emergent Narrative. In: Working Notes of the Narrative Intelligence Symposium (1999)
2. Bryson, J.J., Stein, L.A.: Modularity and design in reactive intelligence. In: International Joint Conference on Artificial Intelligence. vol. 17, pp. 1115–1120. LAWRENCE ERLBAUM ASSOCIATES LTD (2001)

3. Dias, J., Mascarenhas, S., Paiva, A.: Fatima modular: Towards an agent architecture with a generic appraisal framework. In: Emotion modeling, pp. 44–56. Springer (2014)
4. Evans, R., Short, E.: Versu: a simulationist storytelling system. IEEE Transactions on Computational Intelligence and AI in Games **6**(2), 113–130 (2014)
5. Friedhoff, J.: Untangling twine: A platform study. In: DiGRA conference (2013)
6. Grow, A., Gaudl, S.E., Gomes, P., Mateas, M., Wardrip-Fruin, N.: A methodology for requirements analysis of AI architecture authoring tools. In: Foundations of Digital Games (2014)
7. Heckel, F.W.P., Youngblood, G.M., Hale, D.H.: Behaviorshop: An intuitive interface for interactive character design. In: AIIDE (2009)
8. Isla, D.: Handling complexity in halo 2 ai. http://www.gamasutra.com/view/feature/130663/gdc_2005_proceeding_handling_.php (2005)
9. Kapadia, M., Zünd, F., Falk, J., Marti, M., Sumner, R.W., Gross, M.: Evaluating the authoring complexity of interactive narratives with interactive behaviour trees. Foundations of Digital Games (2015)
10. Klimas, C.: Twine. twinery.org (2009)
11. Mateas, M., Stern, A.: A behavior language for story-based believable agents. IEEE Intelligent Systems **17**(4), 39–47 (2002)
12. Mateas, M., Stern, A.: Façade: An experiment in building a fully-realized interactive drama. In: Game developers conference. vol. 2, pp. 4–8 (2003)
13. McCoy, J., Treanor, M., Samuel, B., Reed, A.A., Wardrip-Fruin, N., Mateas, M.: Prom week. In: Proceedings of the International Conference on the Foundations of Digital Games. pp. 235–237. ACM (2012)
14. McCoy, J., Treanor, M., Samuel, B., Tearse, B., Mateas, M., Wardrip-Fruin, N.: Authoring game-based interactive narrative using social games and comme il faut. In: Proceedings of the 4th International Conference & Festival of the Electronic Literature Organization: Archive & Innovate. pp. 1–8. Citeseer (2010)
15. Medler, B., Magerko, B.: Scribe: A tool for authoring event driven interactive drama. In: International Conference on Technologies for Interactive Digital Storytelling and Entertainment. pp. 139–150. Springer (2006)
16. Millington, I., Funge, J.: Artificial intelligence for games. CRC Press (2009)
17. Neufeld, X., Mostaghim, S., Brand, S.: A hybrid approach to planning and execution in dynamic environments through hierarchical task networks and behavior trees. In: Fourteenth Artificial Intelligence and Interactive Digital Entertainment Conference (2018)
18. Pacian, C.: Weird city interloper. The Interactive Fiction Database (https://ifdb.tads.org/viewgame?id=wrt29d4nlm71udll) (2014)
19. Porteous, J., Cavazza, M., Charles, F.: Applying planning to interactive storytelling: Narrative control using state constraints. ACM Transactions on Intelligent Systems and Technology (TIST) **1**(2), 10 (2010)
20. Riedl, M.O., Young, R.M.: Narrative planning: Balancing plot and character. Journal of Artificial Intelligence Research **39**, 217–268 (2010)
21. Samuel, B., Reed, A., Short, E., Heck, S., Robison, B., Wright, L., Soule, T., Treanor, M., McCoy, J., Sullivan, A., et al.: Playable experiences at aiide 2018. In: Fourteenth Artificial Intelligence and Interactive Digital Entertainment Conference (2018)
22. Samuel, B., Reed, A.A., Maddaloni, P., Mateas, M., Wardrip-Fruin, N.: The ensemble engine: Next-generation social physics. In: Proceedings of the Tenth International Conference on the Foundations of Digital Games (FDG 2015). pp. 22–25 (2015)

23. Shoulson, A., Garcia, F.M., Jones, M., Mead, R., Badler, N.I.: Parameterizing behavior trees. In: International Conference on Motion in Games. pp. 144–155. Springer (2011)
24. Smith, A.: Living worlds: the joy of NPC schedules. Rock Paper Shotgun (2016)
25. Stefnisson, I.S., Thue, D.: Mimisbrunnur: Ai-assisted authoring for interactive storytelling. In: Fourteenth Artificial Intelligence and Interactive Digital Entertainment Conference (2018)
26. Szilas, N., Marty, O., Réty, J.H.: Authoring highly generative interactive drama. In: International Conference on Virtual Storytelling. pp. 37–46. Springer (2003)
27. Van Velsen, M.: Narratoria, an authoring suite for digital interactive narrative. In: FLAIRS Conference. pp. 394–395 (2008)