Multimedia Tools and Applications (2023) 82:4671–4708
https://doi.org/10.1007/s11042-022-12307-2

TRACK 4: DIGITAL GAMES, VIRTUAL REALITY, AND AUGMENTED REALITY

NCCollab: collaborative behavior tree authoring in game development

Md. Yousuf Hossain [1] · Loutfouz Zaman [1]

Received: 7 March 2021 / Revised: 11 November 2021 / Accepted: 17 January 2022 /
Published online: 13 April 2022

Abstract
Game development is a collective process in which a variety of different professionals from different backgrounds collaborate together not only by means of conversational interaction but also collaborative participation, one of which is programming. While collaborative and pair programming solutions exist for text-based programming languages, visual programming has not enjoyed as much attention. These solutions would not only address advanced forms of business communication among team members but could find their use in distance learning, which would have been useful during the pandemic. In our work, we propose a solution for collaborative behavioral animation of NPCs using behavior trees through synchronous and asynchronous modes of collaboration. We conducted a user study with 12 moderately skilled game development university students who were placed in groups of two and engaged in joint fixed behavior tree development tasks using the synchronous and asynchronous modes and auxiliary features of live preview, access and restoration of previous states from behavior tree history, conflict resolution, and instant messaging. Participants also completed a control task where no collaboration was involved and auxiliary features were not available. Feedback form Creativity Support Index, a self-developed questionnaire, and a semi-structured interview were collected. Additionally, task completion times were logged. The results indicate that the two collaborative modes provide expected improvement over the control condition. No significant differences were found between the two collaborative modes. However, the semi-structed interview revealed that the synchronous mode could be useful for quick prototyping, while the asynchronous mode – for most other situations.

Keywords Behavior tree · Collaboration · Game AI · Game development · Unity engine

✉ Loutfouz Zaman
loutfouz.zaman@ontariotechu.ca

Md. Yousuf Hossain
yousuffahim8@gmail.com

[1] Ontario Tech University, 2000 Simcoe Street North, Oshawa, Ontario L1G 0C5, Canada

1 Introduction

Today video game development is a collective process in which a variety of professionals from different backgrounds, such as game designers, programmers, QA specialists, come together and collaborate on a common task by constantly communicating with each other. Solutions for the support of distributed collaboration such as project management [3, 56, 59, 69], version control [28], instant messaging [17, 66], and video conferencing [32, 77] currently facilitate this. Being stand-alone, these systems are not aligned with the programming environment, which may result in either substantial distraction or their underuse due to flow interruption.

People are inherently social, requiring to collaborate, coordinate, and communicate with each other, and the diverse range of applications has emerged to enable them to do so in extensive and diverse ways [65]. People often learn effectively when collaborating together, in particular with the help of technology [65]. *Pair programming* is an agile technique in which two developers work together on the same code or project. Pair programming was found beneficial for teaching and engaging learners, see e.g., [67, 78]. Arguably there may exist an increased and urgent need for tools that support collaboration during the pandemic, when the only option to learn for many remains through distance learning. While work has been done for code-based programming, visual pair programming in comparison has not received as much attention for collaboration to this day.

A visual programming language is a programming language [44, 64] where the users create programs by manipulating program elements graphically rather than by writing textual code [42]. Due to its simplicity of use and low learning curve, visual programming has gained popularity among non-programmers for developing game AI [43]. Using node-based programming, applications are developed by connecting "blocks" of self-contained code. Node-based and other visual programming interfaces have become commonplace in popular game development environments such as *Unreal Engine* [75] and *Unity [74]*, as they facilitate and expedite prototyping and development. Both *Unreal* and *Unity* have native support for visual scripting through *Unreal Blueprints* [7] and *Unity*'s recently acquired *Bolt* [8]). Collaborative options in these tools are also available. Work has been done on collaboration for source code [39, 41, 67], and more recently, for visual programming in education with *Blockly* [76]. However, to the best of our knowledge, currently there is no collaborative environment available for behavior trees. In our work we aim to demonstrate that a collaborative visual programming environment for developing game AI using behavior trees is a viable approach.

A *behavior tree* (BT) is a graphical modeling language to model artificial intelligence (AI) in video games and visual simulators. Although other game AI technologies currently exist, such as *Utility AI* [60] for implementing complex non-player character (NPC) behavior, BTs today remain popular and continue to be included in game engines such as, e.g., *Unreal Engine* [75]. Due to their simplicity of use, BTs make developing game and simulation AI accessible even to non-programmers [43], which is arguably one of the main reasons behind their continuing popularity. Furthermore, BTs are recommended if some game designers are not programmers, the conditions governing the behavior are complex, and the NPCs have aspects of behavior in common [46].

In our work, we focus specifically on the development process of collaborative game AI using behavior trees, as we believe it can help to close the gap between the programmers and non-programmers through the use of visual pair programming and also to facilitate distance learning. As our work will demonstrate this also has a potential to facilitate the process of prototyping.

As a solution, we built *NCCollab* (*NodeCanvas Collaboration*), which is an extension to *NodeCanvas* [51] – a visual behavior authoring framework for *Unity (Unity Technologies,* [74]*)*, which enables developers to create advanced AI behaviors and logic. *NCCollab* facilitates collaborative visual programming of game AI between multiple developers. *NCCollab* supports two modes of collaboration: synchronous and asynchronous. In the synchronous mode, multiple users work on the same BT, where the changes are merged between these users in real time. In the asynchronous mode, multiple users can work on the same or different BT and they can merge their changes with other users' changes whenever they are ready. *NCCollab* is an extension to *NodeCanvas*.

The aim of our work is to investigate if collaboration can be done with behavior trees. In doing so we used HCI research methodology, which included requirements gathering, implementation and validation in a mixed-methods user study. More specifically, we wanted to answer the following research question: *how can programming environments for behavior trees be improved with collaboration features for synchronous and asynchronous pair programming?* Specifically, we were interested in this question in the context of game development education, as this is where we believe most of the use cases for this work will be, similar to the example described above [76]. To address the research question, we created *NCCollab* – a node-based behavior authoring framework to create BTs, which features:

- synchronous and asynchronous collaboration modes;
- live preview of other collaborators' BT alternatives;
- BT history, which allows restoring previous states from history;
- a semi-automatic conflict resolution interface to resolve conflicts that may arise during collaboration;
- a one-to-many difference visualization for showing differences between collaborators' BT alternatives;
- an instant messaging system for collaboration.

We then conducted a user study with participants who were sampled mainly from undergraduate and graduate students in game development, who using fixed tasks, compared synchronous and asynchronous collaboration to a control condition with no collaboration.

2 Related work

2.1 Collaboration in programming environments

The use of collaboration is a well-established practice in a variety of fields, including node-based programming. *Collabode* [30] – a web-based Java integrated development environment designed to support close, synchronous collaboration between multiple programmers. Goldman et al. examined the problem of collaborative coding in the context of program compilation errors introduced by other users which make collaboration more difficult and describe an algorithm for error mediated integration of program code. They evaluated this collaboration algorithm and interface on recorded data from previous pilot experiments with *Collabode*, and via a user study with student and professional programmers. They found that *Collabode* offers appreciable benefits over naïve continuous synchronization without regard to errors and over manual version control. *NCCollab* does not directly stop errors from happening

and does not try to solve them. However, *NCCollab* makes sure if there is an error in the tree, the updates are not sent to the server to prevent other users from becoming affected by the error. *Jimbo* [67] is a collaborative IDE with live preview for facilitating distance learning of programming. *Jimbo* integrates text chat, audio discussion, inline discussion and live preview of programming to support better collaboration and communication, bridge gaps and develop mutual understanding between designers and developers. *Jimbo* lacks a formal study to test the collaborative features in a real scenario. To facilitate better communications, *NCCollab* also aims to facilitate this for game developers through the use of instant messaging, live preview of other users' BT, restoring BTs from history, conflict resolution, and by repurposing vanilla *NodeCanvas* features, such as inline commenting through node comments and creating canvas groups. Besides, *NCCollab* was formally evaluated with a control condition using fixed tasks. *AMOEBA* [6]– a tool to support the instrumentation of communication between novice programmers in non-traditional programming environments. Traditional programming approaches involve a programmer working individually manipulating a single piece of code. However, *AMOEBA* was developed and utilized to facilitate collaboration in a learning scenario. A formal study [6] was conducted in a classroom setting between inexperienced middle school and high school programmers using the Intelligent Programming (IPRO) framework. A collaborative programming environment [53] was introduced where two programmers working jointly on the same algorithm and code. Nosek et al. performed a comparative study, which found that student programmers working collaboratively outperformed individual programmers for aspects of performance, which included readability, functionality, and time and satisfaction (as confidence and enjoyment). A novel Any-Time Collaborative Programming Environment (*ATCoPE*) [23] was introduced to combine traditional non-real-time collaborative programming tools and environments seamlessly with emerging collaborative real-time programming techniques. *ATCoPE* enables collaborative programmers to work in software development scenarios and move flexibly according to their needs between different collaboration modes. Fan et al. have proposed a functional design for *ATCoPE*. However, although they validated the feasibility performance evaluation, they did not implement the system and did not run a formal user study. Examples in education include *Blockly* [76] and *Scratch* [64], which use visual programming for collaborative developing interactive 2D applications.

The works above feature both synchronous and asynchronous collaborative systems for code-based programming. As a result, visual programming specific techniques for supporting collaboration are not present in these works. *NCCollab*, in contrast, features access to the history, the ability to revert to previous states, or graphical difference visualizations between collaborators' digital works. *NCCollab* also offers a conflict resolution for solving conflicts during collaboration. This is quite different from traditional programming languages, which are text-based unlike *NCCollab*. Although underneath the tree-based structure there is Json file to store all the information for the BT, the users never interact with that Json file directly. Since the traditional conflict resolutions algorithms are built for code-based programming languages, there is very little to no use of them in regard to a visual scripting, such as *NCCollab*.

2.2 Collaboration in non-programming environments

Research on collaboration is an extensive field. Here, we will only focus on works that support virtual/remote collaboration. The goal of *NCCollab* is to support collaborative game AI authoring in the style of cloud-based services such as Google *Docs* (*Google Docs* [31]),

Microsoft *OneDrive* [47], *Git* [28], Expat Software *Twiddla* [72], *Nulab* (*Online Collaboration Tools for Modern Teams* [54]), and *Cacoo* (*Online Diagram and Flowchart Software* | [55]) because these are the common platforms used for collaboration. These platforms allow addition of files in *Git*, text in Google *Docs* and *OneDrive*, and shapes in *Twiddla* and *Cacco* from multiple users with no conflicts or warnings.

Table 1 presents a comparison between these collaborative tools for operations such as addition, deletion and updating. Flowchart or text-based tools such as *Twiddla*, *Nulab*, and *Cacoo* allow a user to delete or update another user's file without warning. However, version control systems such as *Git* always ensures a user does not delete or modify a file belonging to another user by mistake. For example, a user has pushed their modified file to a remote server while other users are still modifying the previous version of the file before the first user pushes changes. In this case, these other users have inconsistent versions of the file compared to the remote server's version. Now, if they try to push their versions, *Git* will not allow this to prevent conflicts. In *NCCollab* only addition of nodes is allowed without warnings but, when it comes to deletion or updating, the user needs to solve every conflict to merge with another user's BT. Also, in *NCCollab*, similar to *Git*, if the user deletes or makes changes that they are not satisfied with during collaboration, there is an option to restore a previous version of the BT from the history.

2.3 Collaboration in game development

LevelMerge [62] is a collaborative game level editing tool that supports editing scenes and behavior scripts in *Unity* by merging labeled graphs. Using *LevelMerge*, level designers can be designing the game scene, artists can be designing 3D models or its materials, while programmers can work on the scripts at the same time. The system, however, has not been formally evaluated. In contrast, *NCCollab* focuses on a more specific aspect of game development collaboration which is developing game AI. Similar to *LevelMerge* [62], the intrinsic hierarchical structure, graphs are commonly used to represent game levels. *XML3DRepo* [19] is an optimized API for game scenes with versioning and encoding all the objects and assets into a unified game scene graph. The system has been evaluated on the basis of the cumulative CPU decoding

Table 1 A comparison of collaborative tools for common operations

Operations	Google *Doc* (Sync)	*OneDrive* (Async)	*Git* (Async)	*Twiddla* (Sync)	*Cacoo* (Sync)
Add in a different Paragraph (Text)/ Branch (Tree)/ Shape (Diagram)	No warning	No warning	No warning	No warning	No warning
Add in the same Paragraph (Text)/ Branch (Tree)/ Shape (Diagram)	No warning	Paragraph locked while editing.	No warning	No warning	Wait until shape completed, No warning
Update/Overwrite	No warning	Paragraph locked while editing.	Warning to manually fix conflict	No warning	Wait until shape completed, No warning
Delete	No warning	Paragraph locked while editing.	Force Delete	No warning	No warning

time and the overall download time for a variety of 3D models. The system was evaluated by the authors but not research participants.

The industry and academic tools mentioned here allow collaboration between users of different backgrounds. *NCCollab* was designed to support collaboration between programmers or AI designers only. *NCCollab* can also facilitate collaboration between programmers and non-programmers, since one of the key benefits of visual scripting tools is that it can be understood by users with limited programming experience.

2.4 Communication features in collaborative tools

The integration of communication features into collaborative tools can help collaborators discuss and resolve issues without losing focus on the code [67]. For collaborative environments, built-in instant messaging is found to be effective, see e.g., [39, 63, 67].

NCCollab offers a built-in instant messaging system where collaborators can leave notes or ask for each other's help. *NCCollab* follows the "continuous coordination" model introduced by van der Hoek et al. [41] where a system needs to notify the developers of events relevant to them.

2.5 Versioning

NCCollab implements branching, diffing and selective merging for BTs, which are the features commonly associated with version control. Traditionally these techniques have been used for source code and text but, more recently, they have been adapted for more sophisticated types of data. Methods that utilize recording of editing operations have been introduced for 2D images [13] and 3D models [15]. In contrast, a method [12] was introduced that uses the results of editing operations for 3D scenes. Earlier examples include works by Dobos et al. [18–21]. All these works, however, are orthogonal to our work, since we focus on interfaces for AI asset creation. In contrast, *LevelMerge* [62] is a collaborative environment for level editing that supports editing of scenes and behavior scripts in *Unity*, which also supports diffing and merging. Branching, merging, history for node-based generative design has been introduced in *GEM-NI* [81]. All except merging have been found useful in the evaluation. To the best of our knowledge, we are the first to study some of these versioning interfaces on game AI asset creation interfaces.

2.6 Difference visualizations of graphs and trees

Different representation techniques for a single tree, trees in pair and multi-tree visualization have been previously surveyed [33]. Furthermore, a general taxonomy of visual designs that can be used for comparison of three basic categories of trees has been proposed [29]. In these two works the following common techniques were identified: juxtaposition (e.g., [49]), overlay (e.g., [16, 79]), explicit codes (e.g., [80]), animation ([11, 40], matrix, agglomeration and edge drawing.

Side-by-side views is a technique related to our variant of one-to-many diffing. Examples of this technique include *DualNet* [50], which visualizes sub-networks of node-link diagrams with side-by-side views. *TreeJuxtaposer* [49] compares large trees with side-by-side views.

TreeVersity [34–37] shows changes in topology and node values by using glyphs that pre-attentively highlight changes. *TreeVersity* also highlights new and deleted nodes. A new technique for comparing multiple hierarchies on global and local structures for use in biology has been introduced [9]. This work is orthogonal since it targets trees that are significantly more complex than behavior trees. Moreover, our trees encode priority information, which cannot be captured using this approach. *MACE* [80] is an interface for comparing two or more generative node-based model alternatives using a one-to-many diffing interface. In this work, two types of encodings were introduced for diffing in the model viewer and both were found effective.

Table 2 summarizes the works that influenced the diffing techniques implemented in *NCCollab*. In *NCCollab,* we employ explicit codes, a variant of change highlights, juxtaposition and a variant of overlay for showing deleted nodes. *NCCollab* also builds on the additive encoding and a one-to-many diffing technique of *MACE* [80], by introducing an additional encoding for priority change, which is unique to BTs. The user can compare their BT with that of the collaborator using a variant of side views. *NCCollab* uses the top-down layout as it is an easier layout for visualizing common BT systems in game development tools, such as *Unreal Engine's Behavior Trees* [5], *Behavior Designer* [4], etc. It is also important to note that the work behind *NCCollab* is related to the concept of a *modeling language*. A significant body of work has been done in the area of model-driven engineering. This includes collaborative modeling [61], model differencing, patching and merging tools [1, 45].

3 NODECANVAS collaboration

Here we present *NCCollab* (*NodeCanvas Collaboration*) – an extension to *NodeCanvas* that supports collaboration between game developers so that they can improve the adaptation of unplanned challenges and help co-developers when they are stuck. The development of *NCCollab* was preceded by requirements gathering from self-identified practicing game developers on online forums. This helped us to arrive at major design decisions. This is presented below and followed by the system description using worked examples.

3.1 Pre-development requirements gathering

Before development on *NCCollab* started, we wanted to solicit data for our requirements from potential future users. We posted a list of questions online on Unity Forum [73], GameDev Forum [26], Dream.in.code [58], TIGForums [52, 71]. The list consisted of the following questions:

1. Which tool or resources do you use while developing games collaboratively?

Table 2 Relevant diffing techniques in *NCCollab*

	Explicit codes	Overlay	Juxtaposition
Works	[80]	[16,79]	[49] [50] [36] [37] [35] [34] [80]
NCCollab	additive encoding, priority change	deleted nodes	side views to compare BTs between collaborators, one-to-many diffing

2. Do you think that using collaboration features in a visual scripting tool (such as *Unreal Blueprints*, *Playmaker*, *Bolt*, *NodeCanvas*) for game development would benefit you during game development? If so, could you please explain how?
3. Tracking history allows access to previous states quickly. Do you think having this feature while developing games collaboratively using a visual scripting tool would be useful? Why? If so, then how?
4. Do you think the ability to perform collaborative game development in real-time using a visual scripting tool would be useful (e.g., real-time collaborative editing in Google Docs)? If so, then how? Or do you think asynchronous collaborative visual scripting would be more useful? Why?

3.2 Results of pre-development requirements gathering

In total, we managed to solicit data from a total of eight respondents online. The findings were as follows:

3.2.1 Question 1

All the solicited respondents stated that for collaborative development they used *Git*. This included also those who had experience with only scripting languages.

3.2.2 Question 2

One respondent stated: '*Providing an easy-to-understand form of scripting for 2+ users to work within real-time is already a step up from what Unreal Blueprint system provides (no merging, local only). Such a tool would make working with a non-programmer much easier and debugging projects as a group becomes much more streamlined*'. Another respondent stated: '*collaboration will be helpful in scenarios where I am working with another team member – such as an artist or designer – who is not as well-versed in the game's code. In cases like this, being able to edit the script together can be helpful. With everything being online right now, collaborative tools can be very helpful to communicate across development teams*'. Two other respondents mentioned that it will be useful for prototyping ideas with other people, instead of being stuck in an echo chamber where they are on their own and unsure of what decisions to make.

3.2.3 Question 3

One respondent stated: '*Tracking history would be extremely helpful for bug checking for example, if you have a previous working version you can easily backtrack to that working version*'. Besides, three respondents expressed that they were confused an alternative history tracking mechanism is needed, as according to them, *Git*'s history already does the work. We tried to address this by designing BT history in such a way so that it keeps track of all the changes of the BT while *Git*'s history tracks changes for every push to the server. Also, BT history keeps track of changes automatically without interfering with the development flow, while with *Git*'s history

explicit pushes need to be initiated by the user who decides which significant events need to go on record. While this offers more customization of history, *Git*'s process is arguably more interrupting to the development flow.

3.2.4 Question 4

7 of 8 respondents replied to this from a point of view of text-based programming. One respondent stated they always prefer the asynchronous mode for text-based programming but would like to try the synchronous mode for visual scripting. Another respondent stated that '*Synchronous mode would be nice for fast prototyping because two participants can put some concrete idea really fast*'. Yet another respondent stated that it would be beneficial to make sure there are no compilation errors before syncing with other users – something *Git*, being language and syntax neutral, does not inherently support. This feature was implemented in *NCCollab*.

3.2.5 Summary and implications for interaction design in NCCollab

The responses we received helped us to expand our initial thoughts on a collaborative visual environment for behavioral NPC animations.

From the responses to the first question, we identified that *Git* is used by the programmers for collaboration. Although this is a common knowledge, while developing *NCCollab*, we tried not to significantly deviate from how *Git* is used in collaboration.

From the responses to Question 2, we identified use case scenarios for *NCCollab*. Among these is a situation where an artist or designer who are not proficient in programming could edit the scripts together.

From the responses to Question 3, we identified that BT history which allows access to previous states of BTs is different from traditional *Git* history. This is because history tracking in *NCCollab* is more non-intrusive compared to *Git*, since *Git* users need to push the code to the server to keep track of history. In contrast, *NCCollab* does this in the background without interfering with the flow of BT development.

From the responses to Question 4, we identified that both synchronous and asynchronous collaboration could be useful. We also identified that the preference will vary between different users and use cases. As a result, both modes were implemented in *NCCollab* and later evaluated in a user study described below.

3.3 Introduction of *NodeCanvas* collaboration

Here we present *NCCollab* (*NodeCanvas* Collaboration) – an extension to *NodeCanvas* that supports collaboration between game developers so that they can improve the adaptation of unplanned challenges and help co-developers when they are stuck. In *NCCollab*, developers can collaborate both synchronously and asynchronously while working on BTs. In order to add collaboration features to a visual programming environment, we need to achieve two goals by identifying:

1. The concepts that need to be conveyed in these channels,
2. The UIs to use to implement these features?

To achieve the first goal, we have drawn on several concepts from the previous work and applied them to visual programming. As a result, the following forms of interactions were implemented:

a) two types of collaboration (synchronous and asynchronous);
b) a live preview of other collaborators' BTs and a build-in text messaging option to chat with other collaborators;
c) ability to track and restore BTs from history;
d) conflict resolution.

To achieve the goal, we strived to keep the balance between delivering a variety of functionalities and not overwhelming the user with complex interaction, since this would defeat the purpose of using visual programming in the first place. To accomplish this, we aligned implementing all the auxiliary features with collaboration, which was the focus in the development of *NCCollab*. *NCCollab* was developed on top of *NodeCanvas* by modifying the source code of *NodeCanvas* [52]. On the server side, *NCCollab* uses Google Firebase [25]. As a result, there could be a practical limit on the number of collaborators working at the same time due to the limit on the number of responses to the API requests at a time but we do not foresee this to happen often with normal use. Moreover, *NCCollab* makes sure that it does not sync with the other user's BT if there is an error in the BT or game source code.

3.3.1 Support for alternatives

NCCollab supports collaborative exploration of multiple BT alternatives. However, we chose not to evaluate this feature in our user study as it is beyond the scope of the research goals we set for this work. The support for alternatives we implemented in *NodeCanvas* and their evaluation is to be described in a separate publication. However, references to this feature will appear in this work due to its tight integration with *NCCollab*.

3.4 Asynchronous collaboration

In *NCCollab*, the user can collaborate with others asynchronously. In this mode, the users work on their own BT. At any time, if desired, they can sync with another collaborator's BT. This is done by clicking on the "Sync" button in the *NCCollab* toolbar. See, e.g., Fig. 5. The user has the option to work on the same BT, a BT alternative of the same game object, or a different game object altogether.

When our version of *NodeCanvas* is launched *NCCollab* is inactive by default. To enable asynchronous collaboration, the user must check the "Asynchronous Collaboration Mode" under the "Prefs" tab as shown in Fig. 1.

When the user enables asynchronous collaboration, *NCCollab* starts sending all the recent BT data (listed below) to the Google Firebase real-time database. The data is saved as a JSON file in the Google *Firebase*. To simplify storage in the database and minimize the number of API requests, three pieces of data are transmitted to the server, which are:

- the current BT JSON file that is hosted in the canvas allowing other users to sync their BTs with;
- a JSON file that consists of a unique ID (*AltId*) for all the nodes that have been created by the users to keep track of which nodes were created by which users, and
- a Boolean variable that sends an update to other users that there has been a change made on this BT or its alternative (if there are multiple of them) by a specific user.

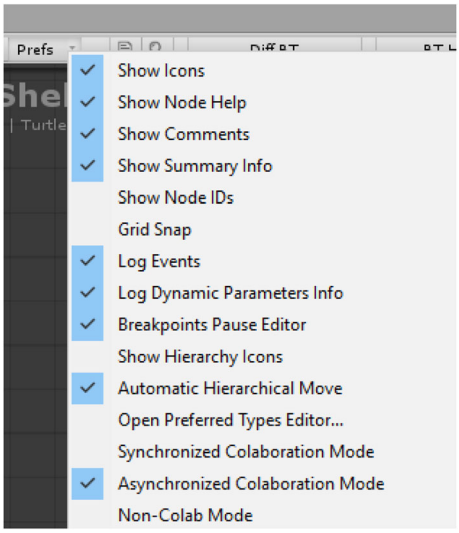

Fig. 1 Enabling Asynchronous Collaboration Mode in the Prefs tab menu

A fundamental requirement to support collaboration is awareness. Dourish and Belloti define awareness as '*an understanding of the activities of others that provides a context for your own activity*' [22]. To support the requirement for awareness in a collaborative system, it needs to provide information about development activities or to notify developers of events relevant to them, such as tree changes, comments, etc. To notify about the changes made in a BT, *NCCollab* uses a refresh icon appearing to the right of the alternative's name in the BT selector window (Fig. 2). When the users hover their mouse above the alternative's name or the update icon, they can see who has updated the BT as the name of the user appears in the tooltip.

Imagine, two game developers John and Sarah, are trying to create an AI behavior for a game object named *TurtleShell* in a game jam working from home. Figure 3 shows *TurtleShell* in the game scene chasing the player. The corresponding BT is at the bottom. The BT Selector panel is on the bottom left. John and Sarah have an overall idea that their AI character will have three states patrolling, attacking, and fleeing. They decided that John will be working on the patrolling state and Sarah will be working the attacking and fleeing states.

John starts working on the patrolling state. In his vision, the AI character moves from one position to another and after arriving at each position the character rests for 2 s. To perform these actions, John creates a selector node, a binary selector node (If Can See $Player), a sequencer node, and four action nodes. John then assigns necessary tasks to the nodes as

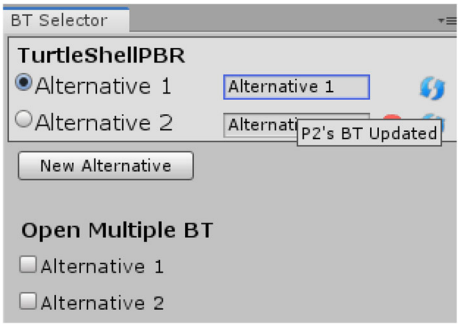

Fig. 2 BT Selector Window with Update Icon

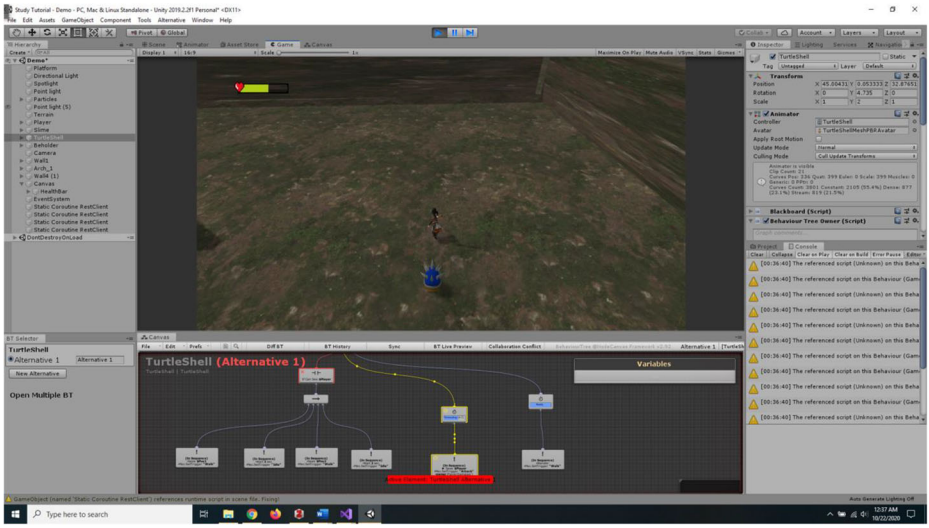

Fig. 3 Overview of the *NCCollab* interface

shown in Fig. 4(a). *NCCollab* sends the BT data to the cloud every time an update has been made by John except for if there is an error in the game. Since Sarah is also using *NCCollab* in the asynchronous mode, these changes will not be automatically applied to her BT. However,

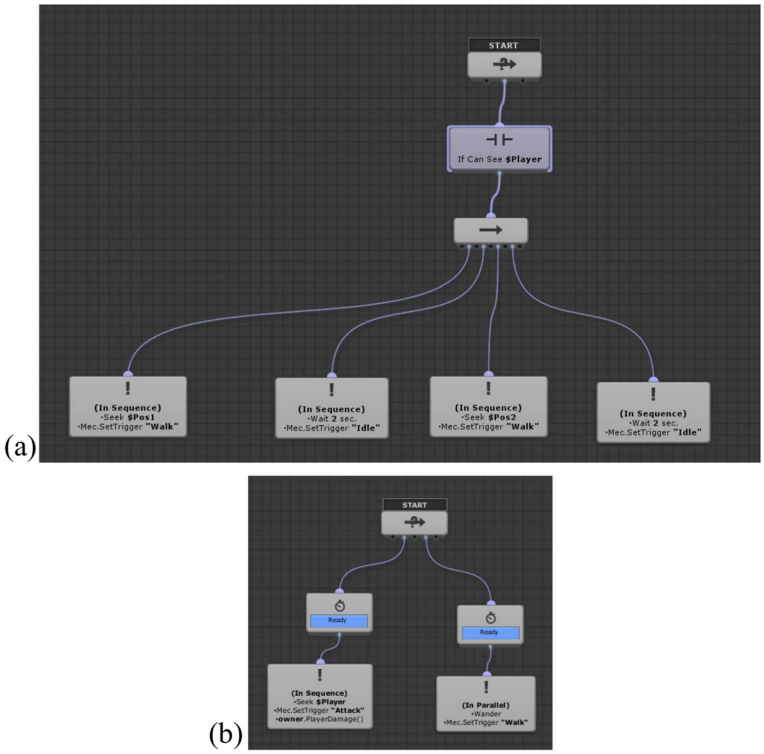

Fig. 4 BT of the *TurtleShell* AI Character: (**a**) John's, (**b**) Sarah's

she can see in the BT selector panel that an update has been made by John and she can also check the live preview of John's BT in another canvas, which is discussed below.

At the same time, Sarah is also working on the attacking and fleeing states in the *TurtleShell* (AI character) BT. In the attacking state, the AI character starts chasing the player. If the AI character reaches the player, the AI character engages in an attack for 15 s. After that, the AI character changes its state to fleeing. In the fleeing state, the AI character wanders around for 15 s. Then the AI character changes its state to patrolling. To perform these actions, Sarah creates a selector node, two timeout nodes, two action nodes, and assigns necessary tasks to the nodes as shown in Fig. 4(b). After finishing the work, Sarah decides to sync her BT with John's. She clicks on the Sync button from the canvas toolbar (Fig. 5). Now, *NCCollab* automatically syncs Sarah's BT with John's. The newly added nodes after syncing are highlighted using a purple outline as shown in Fig. 5. Highlighting allows the user to identify the changes after syncing.

3.5 Synchronous collaboration

In *NCCollab*, the users can also collaborate with other users synchronously. The mode is called synchronous due to the fact that multiple users can work on the same BT at the same time. In synchronous collaboration, each user has the whole working tree locally stored. This allows them to work independently. The network is used to transfer only user-to-user changes. Another reason for storing the trees locally is to prevent users from deleting someone else's nodes to prevent potential errors. If one user deletes, moves or changes nodes, the other users can see them in the collaboration conflicts window (discussed below). To enable synchronous mode, the user needs to check the "Synchronized Collaboration Mode" button in the "Prefs" tab menu (Fig. 1).

Imagine John and Sarah are creating another AI behavior for a game object called *Slime* in a game development jam working from home. Similar to *TurtleShell* (Fig. 3), *Slime* also chases the player. They have an overall idea about what their AI character should be. Like earlier, this

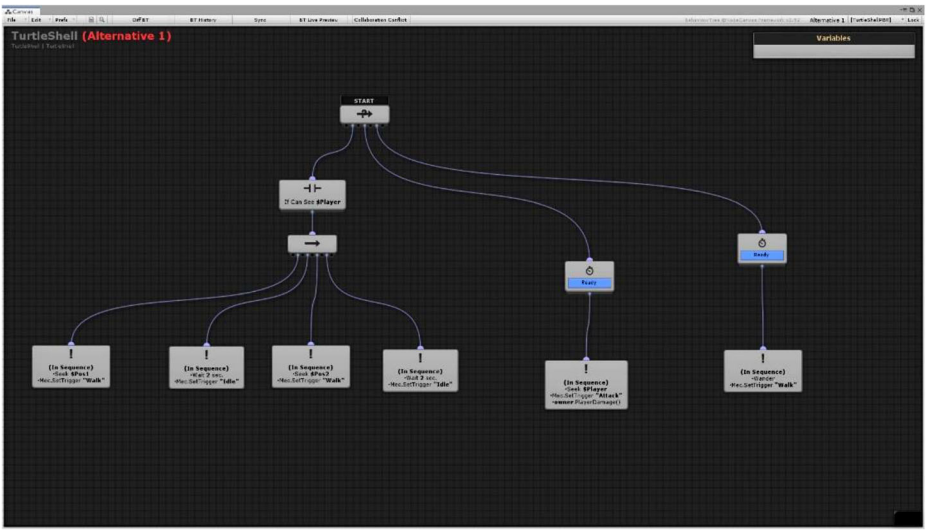

Fig. 5 Sarah's *TurtleShell* BT after syncing with John's BT

AI character has the same distinct states: patrolling, attacking and fleeing. However, how the AI character changes its state is different from before as follows:

1. If the guard (*Slime*) sees the player, the guard attacks.
2. If the guarded character (*Beholder*) is asking for help, the guard attacks.
3. If the guard is attacking but no longer seeing the opponent, the guard goes back to patrolling.
4. If the guard is attacking but is badly hurt, the guard starts fleeing to its home position to regain its full health.

In Fig. 6, a white rectangular outline and user's name are used to differentiate the nodes that were added by John and Sarah (Note: they do not actually appear in the *NCCollab* UI, we added them to the figure help understand the example better). John first starts with the third condition from the above. To perform these actions, he adds a selector node (→), an interrupt node, an action node and assigns all the necessary tasks to these nodes (Fig. 6). In the synchronous mode, all these nodes automatically appear on Sarah's canvas in real time. Sarah starts working on the attacking part of the fourth condition mentioned above. To perform this action, she adds a timeout node. Doing so results in it appearing on John's canvas in real time. Furthermore, Sarah adds a binary selector node (If $Health >5), two action nodes (In Sequence. Seek Player. $Health −=1 Per Second .owner.PlayerDamage() and In Sequence. Seek $Home. $Health = 10). See the "Sarah" group in Fig. 6. While this is being done, John is also working on the third branch of this BT (see the "John" group on the right in Fig. 6) where he completes the second condition from the list above. When everything is done, the resultant BT appears in Fig. 6.

Sarah then decides to make some changes in the patrolling state to enable walking and idle animations. So, she deletes the action node (In Sequence, Seek $Pos1, Seek, Wait 2 s, $Pos2, Wait 2 s). In its place, Sarah adds a sequencer node and four action nodes and assigns necessary action tasks to them as shown using a highlighted white outline with the letter "Sarah" in Fig. 7. Sarah could make changes to her BT. However, these changes do not affect John's BT because *NCCollab* was not designed to allow deletion of other users' nodes. This was implemented to prevent potential loss of important work of other users (only added nodes appear in other users' BTs). John can see these changes in the collaboration conflicts window and the difference visualization view (both discussed below). If John is not happy with the

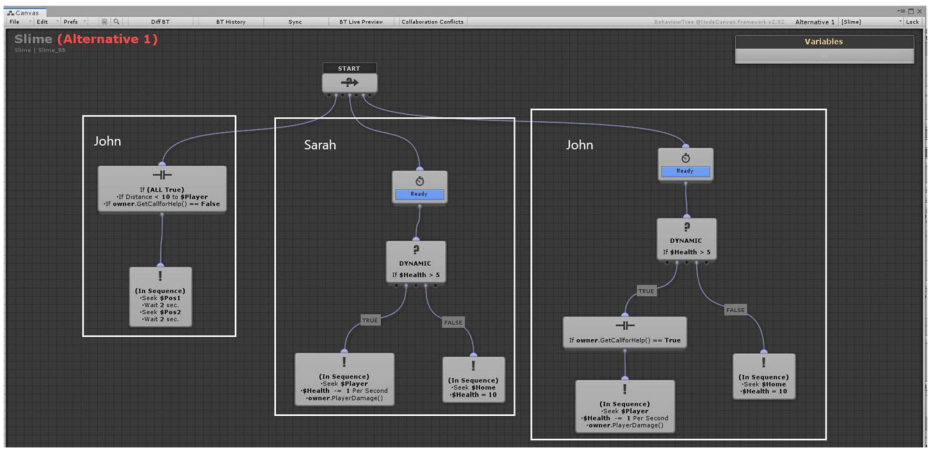

Fig. 6 Resultant *Slime* BT after syncing John and Sarah

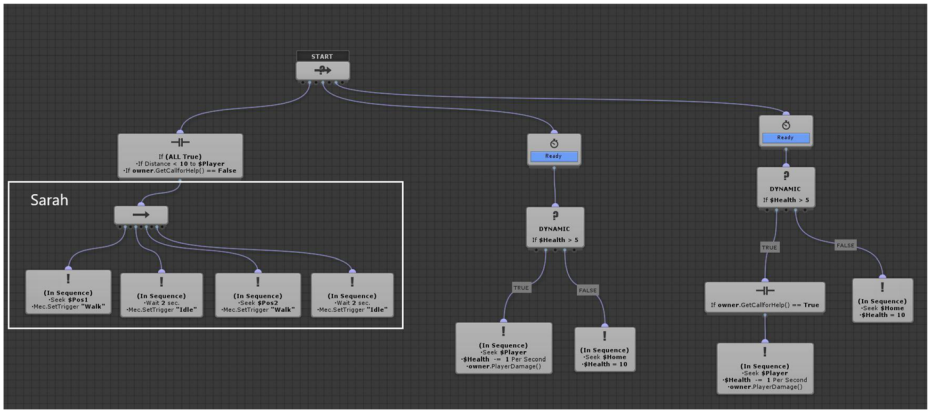

Fig. 7 Sarah's *Slime* BT after enabling walking and idle animation

changes, he can use the "BT history" window (discussed below) and restore from previous states of the nodes from the history.

3.6 Live preview of BT canvas

Live programming is a technique where programmers re-execute a program continuously while editing [68]. Live preview is a variation of this technique that refreshes the output immediately upon a change to the code, and it is ideally suited for UI-heavy application development such as visual programming. In a visual programming environment, this provides an immediate connection between the program's logic and the output for the developers so they can see the effects of changes to their tree or node in real time. This feature leads to fewer iterations of the AI development cycle and expedites the development process overall.

NCCollab provides a live preview of other collaborators' BTs in another canvas (Fig. 8). Using our worked example, John is the remote user. In Fig. 8 John's BT appears on top and its

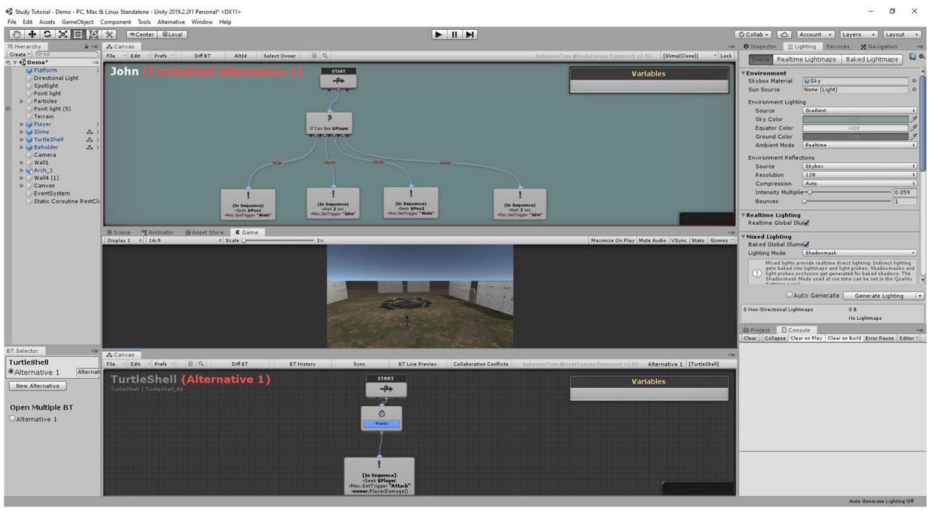

Fig. 8 A live preview of John's BT of the *TurtleShell* AI character from the perspective of the collaborator

background is shaded in dark green to distinguish it from the local version. The collaborator's (Sarah's) own version of the BT appears at the bottom and shows the state when the first part of the BT was complete. The game scene appears in the middle. The panels can be re-arranged in any way the user desires. The game scene always shows the behavior of the local BT (the one the bottom). In this example, John has created a functioning BT, while the collaborator has not started on their BT yet.

Live preview helps the user to have a clear idea of what the other collaborators are working on. This also allows the user to guide other collaborators who are stuck by showing a live preview of their own BT. A live preview of John's *TurtleShell* BT is shown in Fig. 9. To open this live preview, the user needs to click on "BT Live Preview" from the main canvas toolbar (Fig. 5). By default, when the user opens the BT live preview, *NCCollab* automatically re-focuses on the center of the whole graph. This has been done by constructing a rectangle that encapsulates all the positions of the nodes. The rectangle consists of the following points: xMin, xMax, yMin, and yMax. Here, the position of the left most node is xMin, while the position of the right most node is xMax. Similarly, yMin is the position of the lowest node and yMax is the position of the top most node in the canvas. This calculation will be only applied when opening live preview window. After that, the user can move to any part of the tree by holding the left button of the mouse. The background of the canvas color appears green instead of grey to allow the user better differentiate live preview from their working canvas. *NCCollab* shows the user's name, AI character's name and the name of the BT alternative on the top left of the live preview to give a clear idea about the opened BT. The live preview shows what John's canvas looks like in real time except for the zoom and pane of the graph and the positions of the nodes. Since the users can resize the live preview window, changing zoom and pan can be necessary depending on the size of the window.

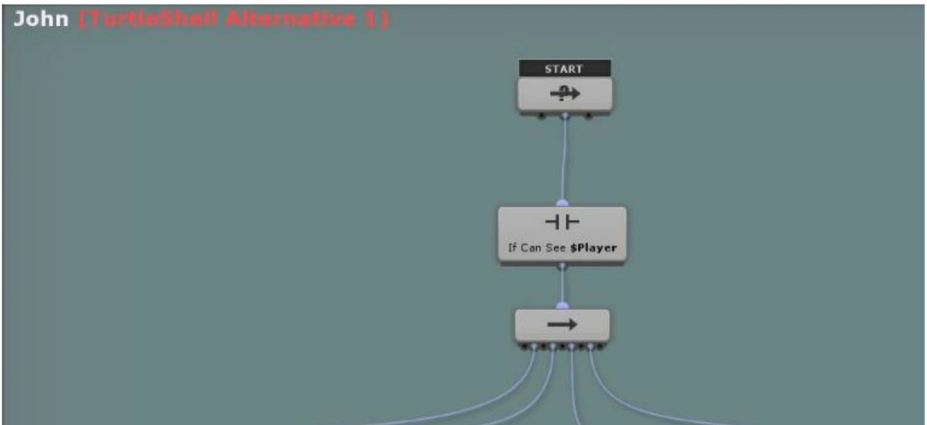

Fig. 9 Live Preview of John's BT of the *TurtleShell* AI Character

3.7 Conflict resolution

In *NCCollab*, conflicts can arise when multiple users modify the properties of the same node or when operations are applied by one user on a node which was deleted by another user or in a situation when a user adds nodes that were never added by another user (only applicable for asynchronous collaboration). In these situations, the user can resolve the conflicts manually, similar to how it is done in version control systems like *Git* [28]. To see the conflicts the user needs to click on the Conflict Resolution GUI button from the *NCCollab* toolbar as shown in Fig. 5. A conflict resolution window then appears where the information is displayed about the type of conflict, who it was caused by, the timestamp, and the details. See Fig. 13.

Imagine, John and Sarah, are creating another AI behavior for a game object named *Beholder* in a game development jam while working from home. *Beholder* is guarded by *Slime* that was described earlier (see Section 3.5 above). *Beholder* has three states based on its health:

1. Idle State: *Beholder* remains in the idle state when its HEALTH >5. If the player attacks the *Beholder*, it asks for help from *Slime*.
2. Dizzy state: *Beholder* stays in the dizzy state when the HEALTH <= 5, and it flees away.
3. Dead State: If the health of *Beholder* is below 0, it dies.

Suppose, John has completed developing all three states of *Beholder*'s BT (Fig. 10). When Sarah syncs with John's BT only the left branch was completed, so she does not have access to the entire BT of *Beholder*. Sarah's *Beholder* BT is shown in Fig. 11. After that, Sarah deleted one action node ($LIFE + = .5 PER SECOND. MEC.SETTRIGGER "IDLE") from the left branch (white rectangle in Fig. 11) and replaced it with another action node ($LIFE + = .5 PER SECOND SEEK POS1. SEEK HOME) as shown in the white rectangle in Fig. 12. To see if there is a conflict, John opens the "Collaboration Conflicts" window as shown in Fig. 13. He finds out that two binary selectors nodes, two action nodes, one condition evaluator node (white rectangle in Fig. 10)

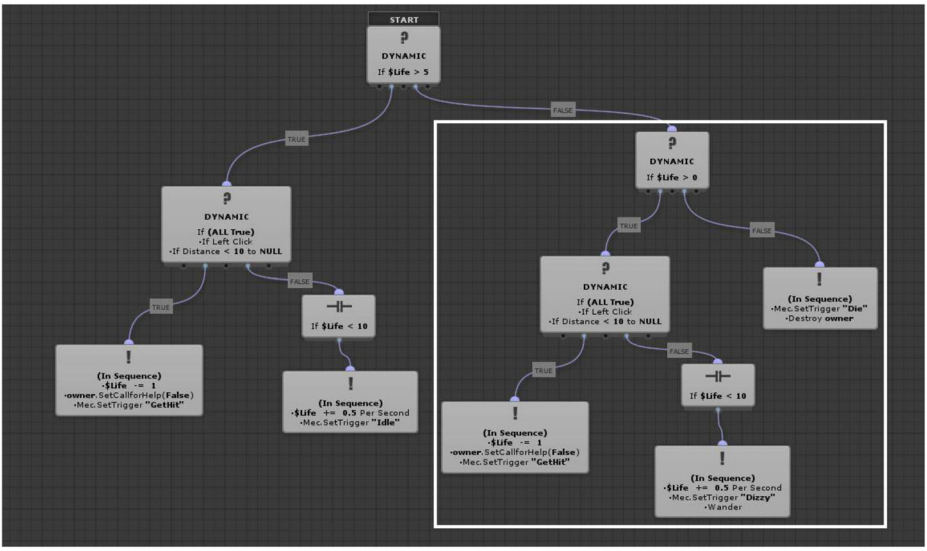

Fig. 10 John's *Beholder* BT with both states completed

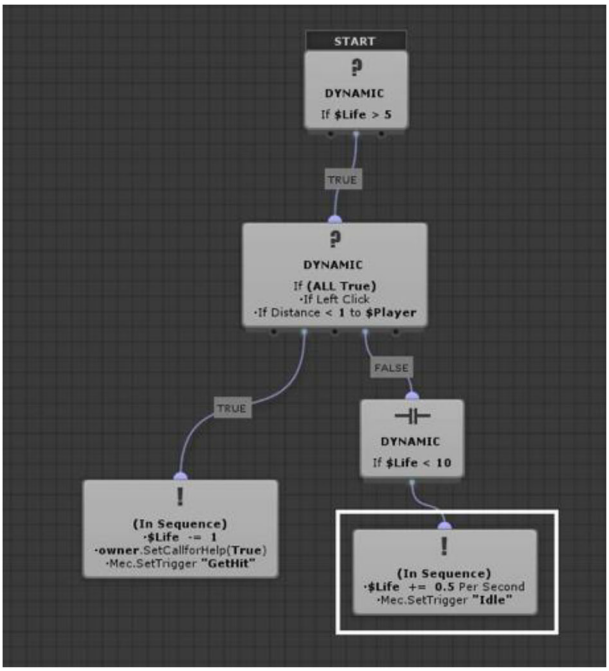

Fig. 11 Sarah's *Beholder* BT with John's left branch only

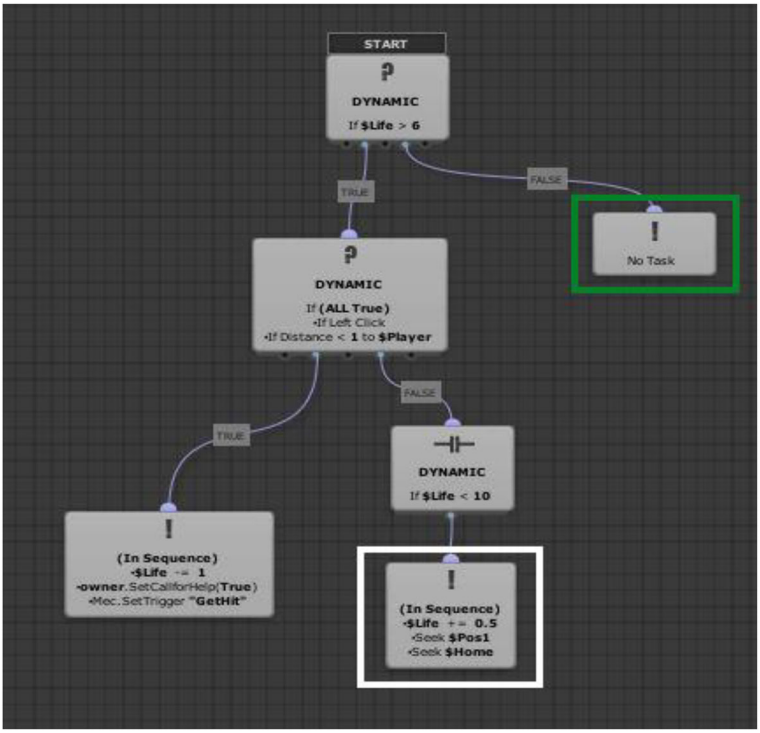

Fig. 12 Sarah's *Beholder* BT with added (green rectangle) and replaced (white rectangle) node

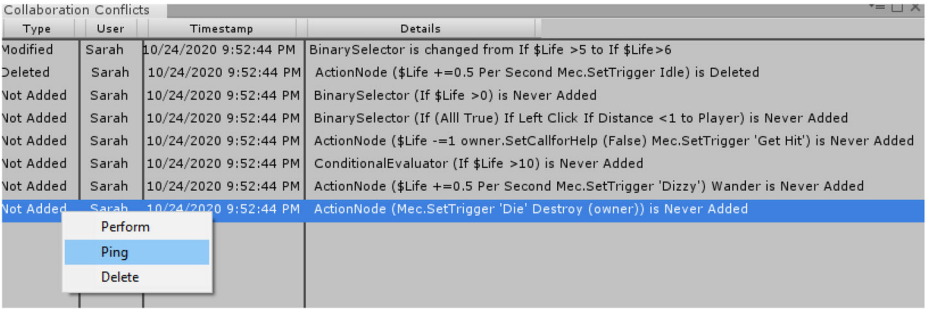

Fig. 13 John's Collaboration Conflicts window using ping

have not been added by Sarah and one action node (In Sequence $Life = −1 Per Second Mac.SetTrigger "Idle") has been deleted by Sarah and replaced with another action node (white rectangle in Fig. 12).

NCCollab uses a red and green rectangular outline to highlight nodes in the canvas that are involved in a conflict. E.g., to highlight a conflict in the BT, a user needs to right-click on a conflict and select "Ping". *NCCollab* then focuses on that node, highlights it using a red rectangular outline and flickers at a rate of 4 Hz between red and green for 2 s (Fig. 14) to get the attention of the user.

NCCollab offers partially automatic collaboration conflict resolution. It is partially automatic because the users do not need to manually modify or delete a node. We call it partial because the users need to choose manually which conflict they want to solve and automatic because they can select "Perform" to automatically resolve the selected conflicts. For example, in Fig. 10 the prime node of John's *Beholder* BT is "If $Life > 5" and in Fig. 12 Sarah modified the prime node to "If $Life > 6". So, to modify John's *Beholder* BT prime node, he needs to click on the "Perform" button as shown in Fig. 15. *NCCollab* cannot resolve "Not Added" conflicts because one user cannot add nodes on another user's canvas. Although if the users do not want to see the Not Added conflicts, they can delete those conflicts from the list. To delete a conflict from the list of collaboration conflicts, the user needs to right click on a conflict from the list and click the "Delete" button in Fig. 15. This will only remove that particular conflict from the list, the conflict will still remain there in the BT but the user does not see it in the list.

3.8 Difference visualizations of BTs

To show difference visualizations in the BT, we will use the same *Beholder*'s BT example that we have discussed above (see Section 3.7). In that example, John completed both states of the

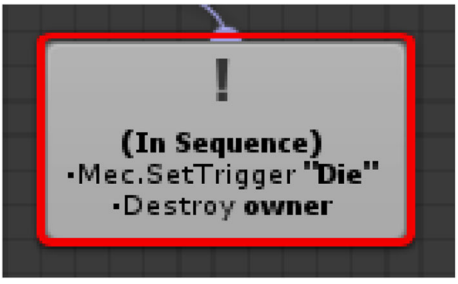

Fig. 14 John's pinged node using Collaboration Conflicts window

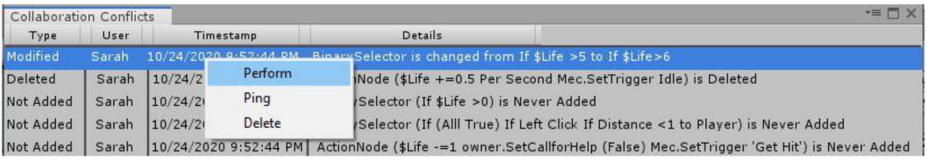

Fig. 15 John selects "Perform" in context sensitive menu in the Collaboration Conflicts window

Beholder's AI character (Fig. 10). However, when Sarah synced with John's BT, only the left branch was completed, therefore she did not have access to the entire BT of *Beholder*'s AI character. Besides, Sarah deleted one action node from the left branch (white rectangle in Fig. 11), added one empty action node (green rectangle in Fig. 12) to the rightmost branch of the tree and modified the first binary selector node (If $Life >6 in Fig. 12). Therefore, when John compares his BT with Sarah's in the BT difference visualizations, he discovers the following differences listed below.

3.8.1 Not added nodes

As shown in Fig. 16, Sarah has six nodes (white rectangle in Fig. 10) relative to John's *Beholder* BT that were not added from before. A missing icon on top of the nodes and reduced transparency informs the user that they have not added these nodes from before.

3.8.2 Deleted nodes

As shown in Fig. 16, Sarah has deleted one node (white rectangle in Fig. 11). A garbage bin icon on top and highlighted as reduced transparency informs the user that they deleted the nodes.

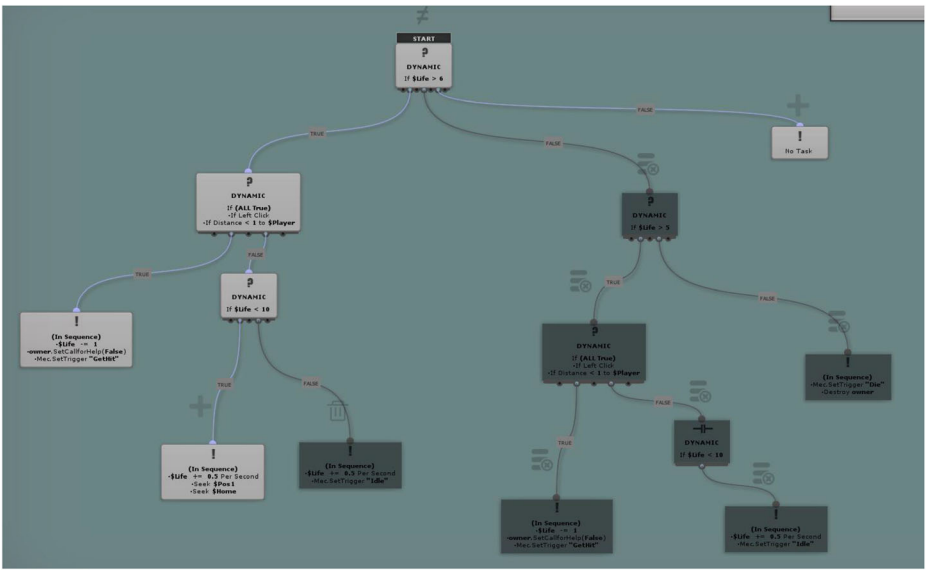

Fig. 16 Difference visualization of the BT of *Beholder* AI Character with differences marked

3.8.3 Added nodes

As shown in Fig. 16, Sarah has added one empty action node. A "+" icon on top of the nodes informs the user that these nodes were newly added relative to John's *Beholder* BT.

Both deleted and not added nodes are visualized with reduced transparency using a complementary shade of grey for the dark green background color of the live preview. This approach of using reduced transparency was found to be effective in the previous work [80].

3.8.4 Modified nodes

As shown in Fig. 16, Sarah has modified one Binary Selector node from $HEALTH >5 to $HEALTH>6. A "≠" icon displaying on top of the nodes informs the user that these nodes were modified relative to John's *Beholder* BT.

3.9 BT history

The ability to restore from a previous state makes *NCCollab* a more complete versioning system by allowing the users to look through their past work and select a particular state of a BT. Previous states are accessible in *NCCollab* in the "BT History" window (Fig. 17a). BT history contains all the previous states of the BT. BT history is updated each time the user performs an operation on the BT. Whenever the user makes a change in the BT such as adding a node or a task, deleting a node or a task, or modifying a node or a task, a new state of the BT appears in the BT History (Fig. 17a). Figure 17b contains the final version 7 of the behavior tree where a sequencer node and two action nodes have been added in the canvas. When the user selects version 4 from BT History (Fig. 18a), version 4 appears in the canvas (Fig. 18b). Version 4 contains a sequencer node and an action node with the "SEEK (0.0, 0.0, 0.0)" assigned task. The user can use the BT History as a version control tool. BT history is stored in a JSON file that contains a unique ID, an operation such as "Added Node Sequencer" or "Added Task Wander", and a filename of a JSON file that contains the entire BT of that version.

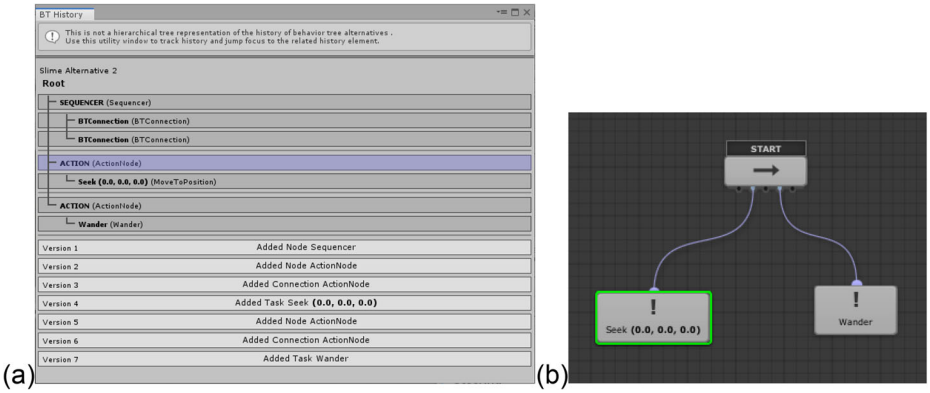

Fig. 17 (**a**) BT History window (**b**) Version 7 opened in the canvas

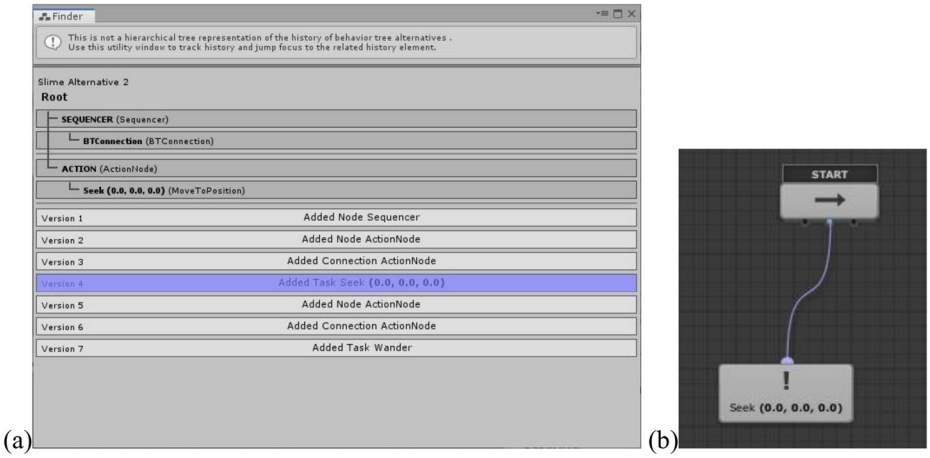

Fig. 18 (**a**) BT History window Version 4 selected (**b**) Version 4 hosted in the canvas

3.10 Hierarchical BT representation

In the BT History window, BTs are shown using a hierarchical view of the tree. This hierarchical representation gives the user visual feedback in the BT History window without the need to look at the canvas for the selected version of the BT every time a change is made. *NCCollab* also helps to visualize a specific node or a connector from the hierarchy by highlighting them in the BT. When the user moves the mouse cursor on top of a node or a connector in the hierarchy, the node or a connect will flicker. In Fig. 17a, the user hovers the mouse cursor above the action node in the hierarchy and in Fig. 17b, the same action node is highlighted in green and flickers.

3.11 Instant messaging

NCCollab offers a built-in instant messaging system where collaborators can leave notes or ask for each other's help. E.g., John is trying to get an update of a BT from Sarah. To start the text chat, John right-clicks on an empty area in the canvas to trigger the context sensitive menu and clicks on "Send message to Sarah "from the list as shown in Fig. 19. Then, the chat box appears (Fig. 20) for the recipient user. Through this chat box John and Sarah can send each other instant text messages. Imagine, John sends Sarah the first message "Hi", but Sarah does not have this chat box open. She receives an instant notification update as shown in Fig. 21.

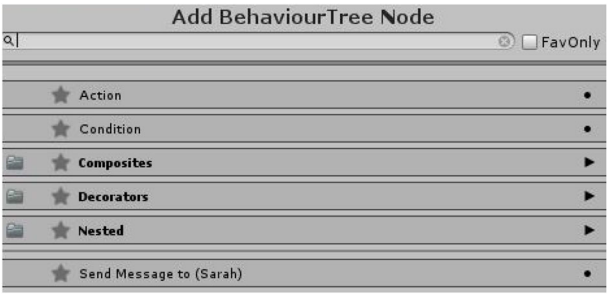

Fig. 19 Accessing the chat box

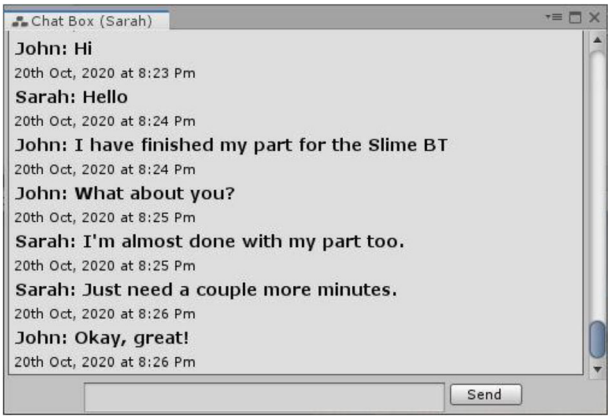

Fig. 20 Accessing the chat window

NCCollab notifies the developers of events relevant to them according the continuous coordination model [41]. *NCCollab* sends an instant notification to the other user because the recipient's chat window was not open at that time.

4 User study

We performed a user study, the goals of which were to:

1. compare synchronous and synchronous modes in *NCCollab* against *NodeCanvas* (the control condition where none of the collaboration support was available);
2. study the usability and usefulness of

a. BT history for restoring previous states of BT,
b. live view of the BT,
c. conflict resolution, and
d. integrated instant messaging;

3. investigate how different collaborative modes affect collaboration, exploration, confidence, chance of future use; and
4. evaluate the overall process.

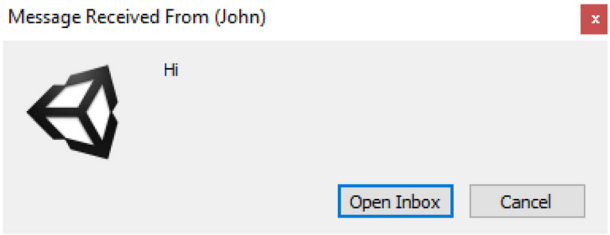

Fig. 21 Message notification window

4.1 Participants

We recruited 12 paid participants (ten males, one female and one preferred not to disclose) from relevant technical undergraduate and graduate programs and from among industry professionals. All are familiar with a concept of game character's AI as all of them had experience creating at least a few games in the course of their studies or as professionals. The participants' backgrounds were game development and computer science. All the participants were between 21 and 35 years old (M = 24.6, SD = 2.6) and had on average 17.3 years of experience using a desktop/laptop computer (SD = 5.3, years). All participants had between 2 and 10 years of experience with *Unity* (M = 5.3, SD = 2.4). Six participants had on average 1.3 years (SD = 1.03) of experience with *Unreal Engine*. All participants on average had 2.1 (SD = 1.8) years of experience with at least one visual scripting system. Seven participants previously worked with *Unreal Engine Blueprints*, three — with *Blender Node Editor*, two — with *Flow Canvas*, two — with *Nodebox,* two — with *NodeCanvas,* one — with *Maya,* one — with *Blackmagic Fusion* and one — with *Nuke*. Some participants also had experience working with block-based system programming languages as follows: four — with *Scratch,* two — with *Blocky,* two — with *Snap!,* one — with *Alice,* and one — with *GML (Game Maker Language)*. We also asked participants to rate their level of proficiency with general-purpose programming languages, such as C, C++, C#, Java, Python, Kotlin, Go, Swift etc. Three participants had average proficiency, six — above average, and another three — professional level proficiency. Two participants took an AI course where they studied behavior trees.

4.2 Apparatus

Since this study was performed online due to COVID-19 pandemic, the workstations and monitors which participants used were a random variable.

4.3 Procedure

4.3.1 Tutorials and tasks

We designed three tasks for the study. One task was performed using *NodeCanvas* (NC), which was the control condition. Another task was performed using the asynchronous mode in *NCCollab*. The third task was performed using the synchronous mode in *NCCollab*. Participants were given a Unity game project template that contained all the game objects and the game scenes for all three tasks. Participants were asked to create an AI for three specific game objects (*Slime*, *TurtleShell*, and *Beholder*). Each of the tasks was preceded by a written tutorial where the corresponding task and template were thoroughly explained. For NC the tutorial covered how to create these different types of nodes and connect them and how to write a script for a node. For *NCCollab* the tutorial also covered how to use synchronous and asynchronous versions of *NCCollab*, versioning of BTs through restoring from BT history window, live BT window to see another participant's canvas, solving conflicts in the behavior tree, and sending text messages through the chat window. The participants were asked to perform a warm-up task as shown in the tutorial so that they are comfortable to use the system when they started the actual tasks.

For the actual tasks, participants were asked to create three behaviors for three game objects in an RPG game. Having three different games for three different tasks would have been ideal, but that would make the study too long as it would require more learning on the part of the participants. So, we decided to use only one game. In *NCCollab*, the participants were only told to use the synchronous or asynchronous modes for two tasks. However, we did not impose the use of the auxiliary features: BT history, live BT window, conflict resolution, and instant messaging. As a result, the participants had full freedom whether they wanted to use the auxiliary features or not. Giving this freedom allowed us to investigate how and in which scenarios the participants would use a certain auxiliary feature in *NCCollab*.

In the control condition (None), the participants were told to create an AI for the game object offline, without the use of collaboration. This also meant they did not communicate with each other and did not coordinate their work in any way. In this condition, they used the vanilla version of *NodeCanvas*. The three tasks involved developing AI behaviors for three AI characters in a game. The tasks were inspired by examples 1 and 2 provided by Richard Moss [48] and by following the guidelines provided by Ben Sizer [70]. The tasks were similar to what we described in the worked example with John and Sarah. All the AI characters' BTs are shown in Figs. 5, 6, and 10. The movement of the humanoid player character was pre-written where the character could:

- move in any direction by using the arrow or WASD keys on the keyboard;
- attack by pressing the left button of the mouse;
- rotate the camera around the player by holding the left mouse button, and
- zoom in and out by scrolling the mouse.

Participants needed to work on the AIs for *Slime*, *TurtleShell*, and *Beholder*. Three functions for the AIs were pre-written in the script. These included *playerdamage*, *SetCallforHelp*, and *CheckCallforHelp*. An empty BT was created for all three AI characters, so the participants had to make their BTs from scratch.

4.3.2 Experimental design, Independent & Dependent Variables, and data collection

The study used a 6 × 3 mixed factorial design. See Table 3 for details.

To minimize the effect of a potential confounding variable of pairing certain tasks with certain modes, the pairings were randomized without replacement according to Table 4.

Methodological triangulation provides multiple perspectives from different data gathering techniques with the goal to validate the results, which has been advocated extensively in the HCI practice, see e.g., [57, 65]. To comply with this practice, we collected data from two questionnaires, data logging and a semi-structured qualitative interview.

Table 3 Independent variables and levels used in the study

Independent Variable	Type	Levels
Order	Between-subject	A. Sync→ Async→None; B. Async→ Sync→ None; C. None → Async→ Sync; D. None → Sync→ Async; E. Async → None → Sync; F. Sync → None → Async.
Mode	Within-subject	None, Async, Sync

Table 4 Randomization of task and mode pairings between participants

Participant #	Task 1	Task 2	Task 3
P1, P2	*Slime* with None	*TurtleShell* with Async	*Beholder* with Sync
P3, P4	*Slime* with Async	*TurtleShell* with Sync	*Beholder* with None
P5, P6	*Slime* with Sync	*TurtleShell* with None	*Beholder* with Async
P7, P8	*Slime* with Sync	*TurtleShell* with Async	*Beholder* with None
P9, P10	*Slime* with None	*TurtleShell* with Sync	*Beholder* with Async
P11, P12	*Slime* with Async	*TurtleShell* with *None*	*Beholder* with Sync

Creativity Support Index (CSI) [14] is a quantitative psychometric survey that measures how well creativity is supported by a tool. In this survey, participants rate six dimensions of creativity support: Enjoyment, Exploration, Expressiveness, Immersion, Results Worth Effort, and Collaboration. We used this survey to measure participants' CSI scores for each mode (None, Async, Sync). This survey is particularly well-suited for rating the extent of how well a certain system supports collaboration. The CSI score was one of the dependent variables.

The CSI was inspired by and modeled after NASA-TLX [38]. Galy et al. [27] propose a method of analyzing the gathered NASA-TLX data, which is to analyze the individual subscales. Analogously, we analyzed individual weighted factor scores from the CSI survey that we administrated. As a result, the weighted factor score was another dependent variable.

4.4 Quantitative results

4.4.1 Creativity support index

The breakdown of CSI results is shown in Table 5.

The main effect of Order was not significant, $F(5,6) = .93$, ns, indicating that counterbalancing was successful. A Mauchly's test for sphericity revealed that sphericity was

Table 5 CSI averages for all three modes

Mode	Factor/ Scale	Collaboration	Enjoyment	Exploration	Expressiveness	Immersion	Results Worth Effort
None	Factor counts (SD)	3.6 (0.7)	1.8 (0.9)	3.5 (0.6)	2.2 (0.8)	1.6 (0.6)	3.8 (1.3)
	Factor Score (SD)	0 (0)	12.9 (1.9)	12.7 (2.3)	13.1 (2.0)	10.7 (1.5)	14.4 (1.7)
	Weighted Factor Score (SD)	0 (0)	22.5 (10.7)	45.6 (17.7)	29.6 (16.4)	17.5 (6.6)	53.0 (15.9)
Async	Factor counts (SD)	3.6 (0.7)	1.8 (0.9)	3.5 (0.6)	2.2 (0.8)	1.6 (0.6)	3.8 (1.3)
	Factor Score (SD)	15.5 (1.5)	14.0 (1.9)	14.3 (1.3)	13.4 (1.6)	10.9 (1.8)	16.1 (0.9)
	Weighted Factor Score (SD)	55.6 (11.1)	23.7 (9.4)	50.4 (13.4)	30.2 (15.7)	18.2 (8.7)	61.2 (20.6)
Sync	Factor counts (SD)	3.6 (0.7)	1.8 (0.9)	3.5 (0.6)	2.2 (0.8)	1.6 (0.6)	3.8 (1.3)
	Factor Score (SD)	13.0 (1.8)	12.8 (2.2)	12.9 (2.1)	12.8 (2.2)	10.9 (2.0)	14.6 (2.4)
	Weighted Factor Score (SD)	46.6 (11.2)	21.6 (9.3)	46.2 (17.1)	28.1 (14.1)	17.1 (7.8)	54.3 (21.1)

violated for Mode, $W = 0.256$, $p < .05$. The degrees of freedom were corrected using Greenhouse-Geisser estimates of sphericity ($\varepsilon = .573$). The main effect of Mode was significant, $F(1.14,6.88) = 26.99$, $p < .001$, $\eta^2_P = .81$. A pairwise post-hoc t-test with Bonferroni correction revealed that the mean CSI score for None ($M = 57.36$, $SD = 11.50$) was different from both: Async ($M = 77.59$, $SD = 8.99$), $p < .0001$ and Sync ($M = 72.27$, $SD = 12.86$), $p < .01$. There was no significant difference between Async and Sync, $p > .05$. See Fig. 22.

4.4.2 Weighted factor scores

We ran mixed ANOVA on weighted factor scores for Enjoyment, Exploration, Expressiveness, Impression, and Results Worth Effort. None of the results were significant. Since collaboration was not rated for None, we ran a dependent t-test, which revealed that Async ($M = 55.5$, $SD = 11.5$) was significantly different from Sync ($M = 46.58$, $SD = 11.66$), $t(11) = 3.5975$, $p < .01$, $d = 0.77$.

4.4.3 Self-developed questionnaire

We asked the participants to rate None as well as Async and Sync modes in *NCCollab* on a Likert scale from 1 (lowest) to 7 (highest) for confidence, efficiency, chance of future use. In Async and Sync modes only, we also asked the participants to rate the features of live BT preview, BT history, conflict resolution, and instant messaging. Finally, all modes were also rated overall. A divergent bar chart summarizes the results in Fig. 23.

A Friedman rank sum test revealed a significant difference for Confidence, $\chi^2(2) = 14.774$, $p < .001$, $W = .615$. A post-hoc Conover test further revealed that Async ($Mdn = 4$) was

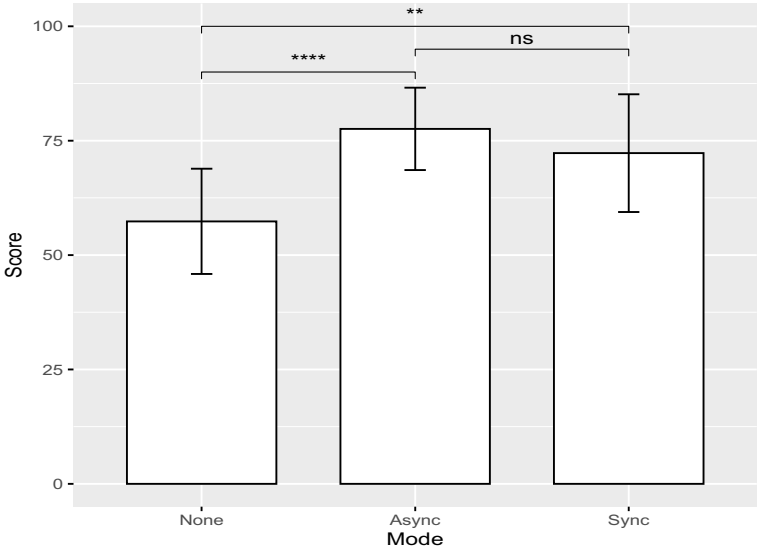

Fig. 22 Mean CSI scores for the two collaboration modes and the control with significance levels from the post-hoc pairwise t-test with Bonferroni correction. Error bars: ±1 *SD*. ns: $p > .05$, **: $p < .01$, ****: $p < .0001$

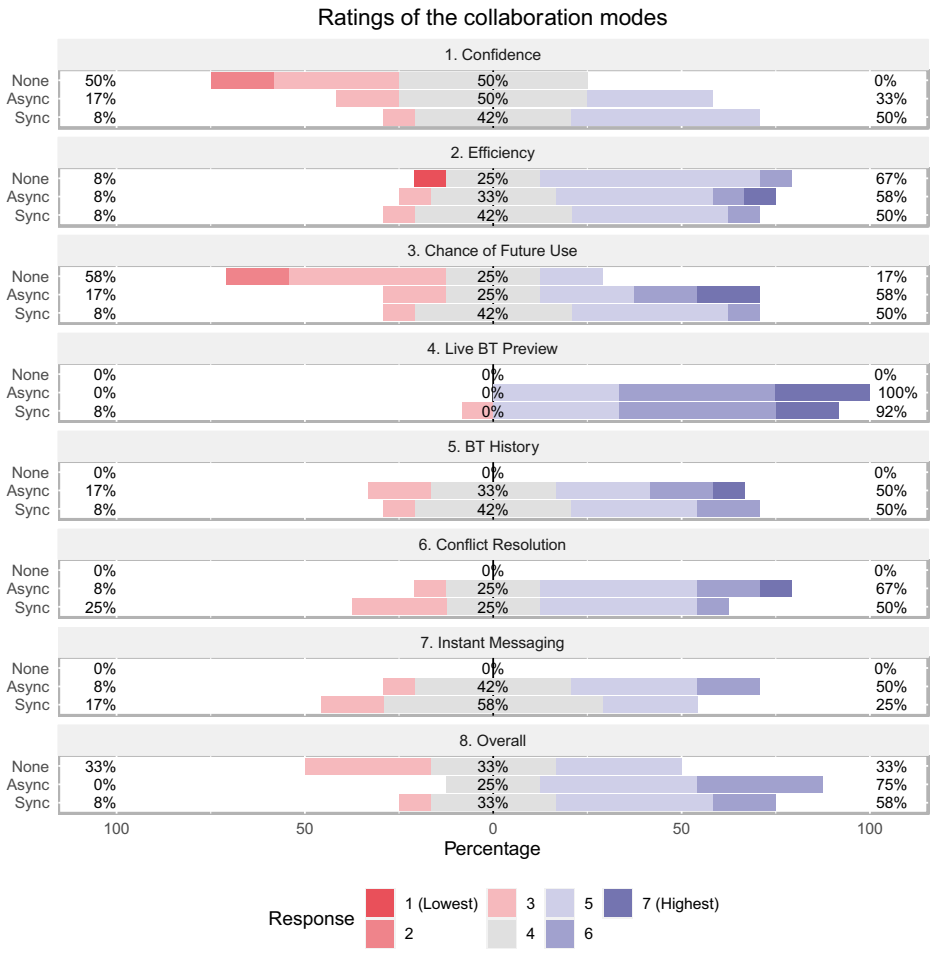

Fig. 23 A diverging stacked bar chart comparing the ratings of the two collaboration modes (Async and Sync) and a None

different from None (Mdn = 3.5), $p < .05$, and that Sync (Mdn = 4.5) was different from None, $p < .05$. A Friedman rank sum test also revealed a significant difference for Chance of Future Use, $\chi^2(2) = 9$, $p < .05$, $W = .375$. A post-hoc Conover test further revealed that Sync (Mdn = 4.5) was different from None (Mdn = 3), $p < .05$. All other differences were not significant.

4.4.4 Task completion time

The main effect of Order was not significant, $F(5,6) = 0.64$, ns, indicating that counterbalancing was successful. The main effect of Mode was significant, $F(2,12) = 5.52$, $p < .05$, $\eta^2_P = .48$. A pairwise post-hoc t-test with Bonferroni correction revealed that the mean task completion time for None (M = 21.32, SD = 3.5) was different from both: Async (M = 16.25, SD = 3.85), $p < .05$ and Sync (M = 17.57, SD = 2.26), $p < .01$. There was no significant difference between Async and Sync, $p > .05$. See Fig. 24.

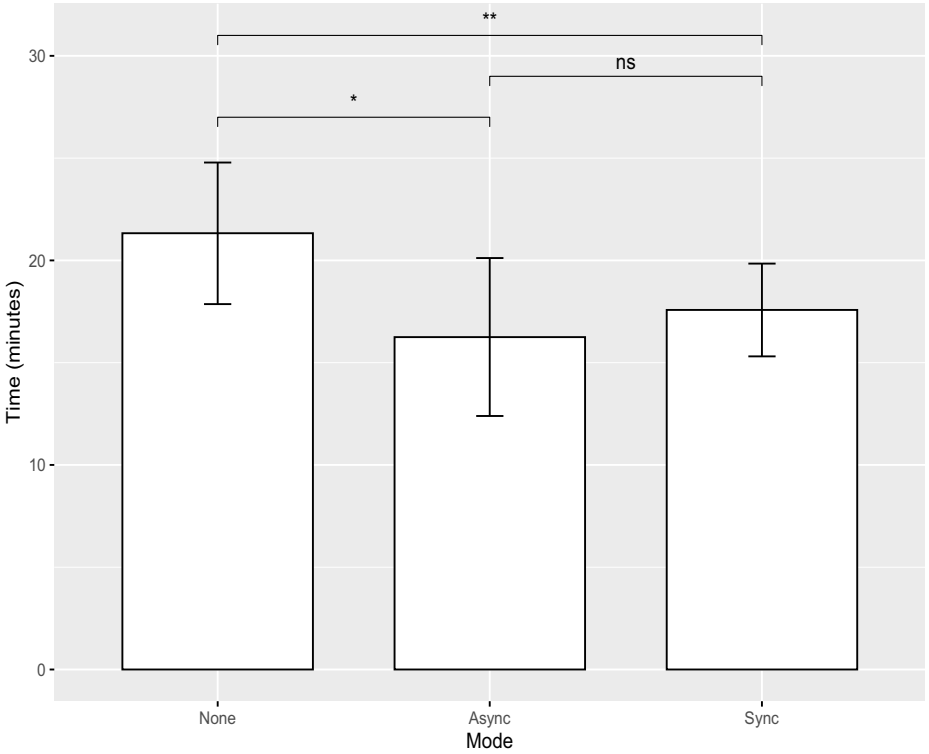

Fig. 24 Mean task completion times for the three collaboration modes with significance levels from the post-hoc pairwise t-test with Bonferroni correction. Error bars: ±1 *SD*. ns: $p > .05$, *: $p < .05$, **: $p < .01$

4.5 Qualitative results

We administered a semi-structured interview with the participants at the end of the study. During this time, the participants were asked to express their opinions, to provide feedback on the overall experience, and to explain the reasoning behind their decisions.

4.5.1 Live preview

Overall, live preview was one of the most popular features of *NCCollab*, about which nearly all the participants felt positive. This is likely because live preview enabled them to be instantly aware about the updates their partners made to their BTs. P4 stated: '*I really enjoyed the live preview feature, being able to see what my peer was working on*'. P2 stated: '*The collaborative features seemed fairly robust, I could preview what was happening over the network and the chat window enhances the collaboration*'. These qualitative findings agree with the results of the self-developed Likert questionnaire in Fig. 23 for live preview. Moreover, P10 suggested that live preview could be also used for teaching and learning.

4.5.2 BT history

Arguably, the addition of BT history brings *NCCollab* one step closer to being a fully-fledged version control system like *Git* [28]. All the participants were familiar with the idea of version

control systems. The participants liked the idea that *NCCollab*, to an extent, offers similar functionality through BT history. P6 stated: *'I like that it's like a live version control. It has lots of analogs to programming and version control making it easier to understand'*. Although P3, P5 mentioned that they did not use the BT history in this study because they were too focused on completing the task, they did see the benefit of having this feature in the long run. They thought that they would be more willing to use BT history if they were working on a project with a larger scope. P5 suggested an interesting use case for BT history: after a long break, if the developers cannot exactly remember which part they were working on, they can just restore BT from history to see the changes they have made before the break. These findings somewhat agree with the results from the self-developed questionnaire in Fig. 23 for BT history, since half felt positive about the feature in both Sync and Async modes, while less than 20% felt negative. P12 did not end up using BT history because they stated that '*I would [be] more willing to use the history tracking feature if I was working on an open-ended task or a larger BT*'.

4.5.3 BT hierarchy

We asked the participants if they found it useful to see a BT represented as a hierarchy. P7 found it useful because they liked simple text-based representation in the form of a hierarchy. However, P12 did not find the hierarchy that beneficial because the original BT structure was already straight forward.

4.5.4 Collaboration

We asked the participants what their preferred mode of collaboration was (None, Async, or Sync). Eight of them were in favor of Async and four were in favor of Sync. P9 stated that they were missing the collaboration features when the vanilla *NodeCanvas* was presented after the two collaborative modes *'I did miss having some of the collaboration features such as the syncing when using this variant. It wasn't necessarily something I disliked but it felt like something was missing without them'*.

4.5.5 Instant messaging

We were observing the instances when the participants used instant messaging during the task. The following purposes of use have been identified as a pattern:

(i) to divide their work on the behavior tree at the start,
(ii) to check if another collaborator can see their update in real time,
(iii) to signal to the other collaborator that the task has been completed or to confirm this.

In one instance, after completing one of the tasks, P7 sent a message to P8 but since *NCCollab* does not use sound to notify the user, P8 never noticed that message. So, P7 had to send multiple messages. P7 stated '*I would want the receiver to get a sound notification after receiving every new message*'. We also asked all the participants opinion on how we could improve the usability of the messaging. P8 stated '*I would prefer voice chat over text chat*'.

4.5.6 Usefulness of NCCollab

We asked the participants regarding the situation where they would use *NCCollab*. P3, P4, P7, P11 stated that since there is a time limit, *NCCollab* would be useful in a game jam. P4 stated '*I would like to use NCCollab's synchronous mode in a game jam where I need to make a prototype of a game in a short time*'. Coincidentally, we have also used this game jam scenario to describe the features of *NCCollab* above. Moreover, P8 mentioned three noteworthy use case scenarios for *NCCollab* as follows:

(i) use by novice users such as design professionals,
(ii) use by two experienced developers,
(iii) use by one experienced and one novice developer.

4.5.7 Game genres

We were interested to learn for developing which types of games *NCCollab* would be most useful according to the participants. P4 suggested it would be useful for prototyping, P9 suggested third-person role-playing games. However, P6 suggested that regardless if collaboration is supported or not, *NodeCanvas* would not work for puzzle games. Although, it is already expected because the main purpose of *NodeCanvas* is to develop the AI of a character, which these genres do not utilize.

4.6 Criticism, Suggestions & Uncovered Issues

We received some useful feedback on how to improve the system further. P6 had an issue with getting lost between two canvas windows because one was the main canvas that they were working on and the other was showing live preview of another user's BT. So, P6 suggested to fit both canvases in one window so that the user can see both at the same time and resize live BT preview. P1 suggested adding a feature of locking a node so that the other users cannot make any changes to them. Moreover, P1 also suggested to have some sort of visual representation to show which node is selected or being edited by another user.

P7 suggested having an optional comment field in the BT history window beside every state, similar to how *Git* [28] has the option to add a comment on a change to provide details. P3 and P12 had a problem with instant messaging, as the messages would not wrap. So, in order to see long messages, they had to make the window wider. P5 also noticed that they could not press the enter key on the keyboard to send a message and was forced to click on the send button, which slowed down their chatting experience.

5 Discussion

Here we describe and interpret the significance of our findings in the light of *NCCollab*.

We used the CSI survey [14] to compare the support of each interaction mode in terms of creativity. Our hypothesis was that the scores will be significantly different between the control condition and the two modes of collaboration, and also between the two collaborative modes. In reality, the results were found significant only in comparison against the control condition,

but not between the collaborative modes. CSI calculates the score by taking into consideration the weighted factor scores for collaboration, enjoyment, exploration, expressiveness, immersion, and results worth effort, as the dimensions of creativity support [14]. The highest possible count for any particular factor is 5. A count of 5 means the factor is more important than others. The maximum factor score is 20. According to Cherry and Latulipe [14], a factor count in the range of 3.6 is an important factor, compared to an average count of 2.5. We found significant differences within the sample that we collected only for collaboration, which is likely the only influential factor contributing to the overall CSI score. This indicates that the participants felt all three modes provide comparable support for creativity, collaboration aside. However, this could be to the fact that the tasks may not have left sufficient room for creativity and/or small sample size, see Section 5.1 (Limitations) below.

A self-developed questionnaire was created to solicit feedback on performance of each auxiliary technique that we implemented to support collaboration when used across the synchronous and asynchronous modes. Additionally, we used this questionnaire to learn how participants felt about each mode in terms of confidence, efficiency, chance of future use, and overall. For confidence, significant differences were found between the control condition and the two collaborative modes. One interpretation could be that the participants felt more confident when working with a partner in contrast to doing solitary work. None of the participants had prior exposure to *NCCollab*, so they were still learning how to use the tool. We speculate that this could be due to the fact that the participants felt like they can rely on help from their partner in case they got stuck for whatever reason, which as a result, could cause less distress. Supporting this, it is worth noting that P8 and P10 suggested that the platform could be useful for teaching and learning due to the availability of live preview. This is in agreement with the vision behind solutions such a *Jimbo* [67]. *NCCollab* can be found useful in educational scenarios with its support of pair programming. For example, the instructors can give some tasks to the students and the students can engage in pair programming and the instructors can monitor their work in the live preview. This is especially useful for distance learning. However, as P8 has mentioned, this does not always need to apply in the situation where the skills of peer programmers differ. According to P8, collaborators with matching skill levels can benefit from this as well.

Unsurprisingly, participants rated vanilla *NodeCanvas* as an unlikely tool for future collaborative use. Interestingly, for chance of future use, significant difference was found between the control and the synchronous modes, but not the asynchronous mode. However, 58% of participants rated the asynchronous mode favorably compared to 50% who did the same for the synchronous mode. Furthermore, the synchronous mode was rated neutral 1.68 times more frequently than the asynchronous mode. These statistical findings aside, we think a broader perspective at the self-developed questionnaire findings suggest that both modes of collaboration have their applications. In particular, P4 mentioned that the synchronous mode would be appropriate for quick prototyping.

During the study, we noted that BT history was utilized by 10 out of 12 participants. P12 was one of the participants who avoided using this feature and justified this due to the task having a small scale. P12 mentioned that instead they relied on their ability to easily keep track of the changes made to the BT in the mind. We speculate larger and more open-ended tasks will end up in more use of the feature.

Three participants wanted to have instant messaging with more advanced features such as voice chat. Other features that were suggested included referring nodes through messaging and wrapping the messages to fit into the screen as mentioned above. In the absence of these

features, it is likely the users will resort to using business communication platforms such as Discord [17] and Slack [66]. We also noticed that all the participants either used conflict resolution or difference visualizations in the tree only after completing each task to check if their own BT fully matched with their partners' BT. 8 out of 12 participants used conflict resolution and four participants chose to use difference visualizations. However, none of the participants complained about difference visualizations, they just preferred to use conflict resolution over difference visualizations. Overall, these findings may indicate that both of these features could be disruptive to the flow, resulting in participants using them after completing their task. This is also supported by the desire to have a voice chat, as text messaging could be disruptive.

Similar to Nosek et al. [53], we logged how long it took the participants to complete the tasks. Both synchronous and asynchronous modes were faster than the control and the order effect was not significant. Our initial hypothesis was that the two collaborative modes would be different. In particular, we thought that in the synchronous mode, since the task was done in tandem, the completion would be faster compared to the asynchronous mode where changes had to be integrated post-hoc at moments deemed appropriate by the participants. We also thought by working in tandem in real time, the participants would develop better strategies to avoid conflicts and coordinate their work better, which would expedite the completion. However, this is not what we found, since there was no significant difference between the collaborative modes. The unsurprising part of the findings was that both collaborative modes were faster than the control, which is typical to expect when the work is shared. This has been also known to be the case for code-based programming for decades [53]. It is worth acknowledging that a limitation in the form of a possibility may exist that the task itself could be a confounding factor since it varied across the levels of the independent variable.

P1 suggested adding a feature of locking a node so that the other users cannot make any changes to them. This is an interesting suggestion. Arguably, it would bring *NCCollab* closer to SVN [2], where users check out files and commit changes back to the server. While a file is checked out, it cannot be modified by other users. This is one of the reasons behind SVN's waning popularity ever since the introduction of *Git* [28].

5.1 Limitations

We made an assumption that a large effect size of $\eta^2_p = 0.14$ or more is likely to be observed. As a result, we targeted a sample size of 16 participants, which should have been sufficient given a minimal statistical power of $1\text{-}\beta = 0.8$ as recommended by Field [24]. In the end, due to the COVID-19 pandemic, the participation of 12 participants is all we were able to secure, which although is lower than needed, still happens to be the most common sample size within the CHI community [10]. Nonetheless, for CSI, the sample size was sufficient to detect the significance. However, the sample size may not have been enough for the analysis of other data that we collected and could be the reason behind multiple insignificant findings.

Another limitation is the fact that the participants have been recruited using convenience sampling, where their willingness to participate was prioritized over our need for random sampling from the population. As a result, generalizations of the results to the population will not be robust [65], p. 261. Most participants were graduate and undergraduate students. While the study results may suggest that the proposed methods are viable approaches for AI game development in education, including distance education, the results, at this state, cannot be confidently generalized to professional game developers. However, it is worth noting that the

solutions we arrived at were informed by the professional game developers. Also, the participants were paid, which could potentially affect the outcomes, although counterbalancing was employed to address this.

At last, due to the COVID-19 pandemic, we performed the study online, which resulted in the variance between the specifications of the PCs which the participants used. Additionally, four participants used a second monitor. As a result, due to our inability to have a full control over the online experimental procedure and the hardware of the participants, there is a possibility that confounding variables may have been introduced.

5.2 Generalizations

It should be noted that all the features of *NCCollab* can be re-implemented in other popular BT editors with minimal changes. The two notable examples are *Behavior Designer* for *Unity* and *Behavior Trees* for *Unreal Engine*. Both of these editors feature top-down and left-to right order of node execution, as well as modifying node properties. As a result, difference visualizations can be directly implemented in them with minimal changes (this also is true for difference visualizations of priory changed nodes). Since both tools feature trees, hierarchical representations can be created for them as well. Likewise, there would not be any obstacles in adapting synchronous and asynchronous collaboration modes, live previous, history visualizations or conflict resolution.

6 Conclusion & future work

Pair programming is common in software development. In recent times due to the COVID-19 pandemic, the demand for the support of pair programming for distance learning, be it text-based or node-based, is what we expect to increase. Game development is a collective process where a variety of different professionals from different backgrounds cooperate. Although solutions for distributed collaboration as standalone applications exist, few are available that consolidate multiple collaborative features for specific tasks like game programming. Furthermore, none, to the best of our knowledge, exist for visual programming and game AI development in particular. As a result, *NCCollab* is our proposed solution which aims to shrink this gap.

Within the limitations of our study, we found that *NCCollab* was received favorably by the participants, and that both modes of interaction can have their legitimate applications. It was suggested that the synchronous collaboration could be useful for quick prototyping, while the asynchronous collaboration could be appropriate for most other forms of collaboration. This addresses our initial research question of how programming environments for behavior trees can be improved with collaboration features for synchronous and asynchronous pair programming, albeit with certain limitations. Some of the results we obtained were insignificant, which we speculate could be partially attributed to the small sample size that was employed in the study. As a result, further research is necessary to make stronger conclusions, which could be pursued in the future work.

In our study, we focused on fixed tasks. We believe this is partially responsible for the fact that significant differences were not found in the analysis of the creativity support of *NCCollab*. In the future, longitudinal and open-ended tasks can be employed to study, e.g., if an interaction effect between the mode of collaboration and creativity support exists.

One participant did not find the hierarchy particularly useful because the behavior tree was straightforward. Future investigations can look into whether this is different in more complex behavior trees. We found that the chat system on its own may not be very useful as the users can use other chat systems. However, future work can look into chat systems that are more integrated with the development tool.

Although the results of our study are applicable to the educational context, they cannot be robustly generalized to professional game development. In the future, studies can be done with professional game developers exclusively to determine applicability of the solutions that we provided.

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1007/s11042-022-12307-2.

Acknowledgements This research was supported by NSERC Discovery.

Declarations

Conflict of interest Authors declare that they have no conflict of interest.

References

1. Alanen, M., & Porres, I. (2003). Difference and Union of Models. In P. Stevens, J. Whittle, & G. Booch (Eds.), *«UML» 2003—The Unified Modeling Language. Modeling Languages and Applications* (Vol. 2863, pp. 2–17). Springer Berlin / Heidelberg. https://doi.org/10.1007/978-3-540-45221-8_2
2. Apache Subversion. (n.d.). Retrieved November 1, 2020, from https://subversion.apache.org/
3. Asana. (n.d.). *Asana Project Management software—online tools, templates & app · asana*. Asana. Retrieved October 27, 2020, from https://asana.com/uses/project-management
4. *Behavior Designer—Behavior Trees for Everyone | Visual Scripting | Unity Asset Store*. (n.d.). Retrieved July 16, 2020, from https://assetstore.unity.com/packages/tools/visual-scripting/behavior-designer-behavior-trees-for-everyone-15277?gclid=Cj0KCQjw9b_4BRCMARIsADMUIypkySxldYUkWqI-HTUpPdYxjzc0Xy23V-YZG8x2T4mSASxIZr5LNmkaAvzIEALw_wcB
5. Behavior Trees. (n.d.). Retrieved August 4, 2020, from https://docs.unrealengine.com/en-US/Engine/ArtificialIntelligence/BehaviorTrees/index.html
6. Berland M, Davis D, Smith CP (2015) AMOEBA: designing for collaboration in computer science classrooms through live learning analytics. Int J Comput-Support Collab Learn 10(4):425–447. https://doi.org/10.1007/s11412-015-9217-z
7. Blueprints Visual Scripting. (n.d.). Retrieved December 22, 2018, from https://docs.unrealengine.com/en-us/Engine/Blueprints
8. *Bolt | Visual Scripting | Unity Asset Store*. (n.d.). Retrieved March 5, 2021, from https://assetstore.unity.com/packages/tools/visual-scripting/bolt-163802
9. Bremm S, von Landesberger T, Heß M, Schreck T, Weil P, Hamacherk K (2011) Interactive visual comparison of multiple trees. IEEE Conf Visual Analytics Sci Technol (VAST) 2011:31–40. https://doi.org/10.1109/VAST.2011.6102439
10. Caine K (2016) Local standards for sample size at CHI. Proceed 2016 CHI Conf Human Factors Comput Syst:981–992. https://doi.org/10.1145/2858036.2858498
11. Card SK, Sun B, Pendleton BA, Heer J, Bodnar JW (2006) Time tree: exploring time changing hierarchies. Visual Anal Sci Technol, 2006 IEEE Symp On:3–10. https://doi.org/10.1109/VAST.2006.261450
12. Carra E, Pellacini F (2019) SceneGit: a practical system for diffing and merging 3D environments. Assoc Comput Mach 38:1–15. https://doi.org/10.1145/3355089.3356550
13. Chen H-T, Wei L-Y, Chang C-F (2011) Nonlinear revision control for images. ACM SIGGRAPH 2011 Papers, 105:1-105:10. https://doi.org/10.1145/1964921.1965000

14. Cherry E, Latulipe C (2014) Quantifying the creativity support of digital tools through the creativity support index. ACM Trans Comput-Human Interaction 21(4):21:1–21:25. https://doi.org/10.1145/2617588
15. Chirigati F, Freire J, Koop D, Silva C (2013) VisTrails provenance traces for benchmarking. Proceed Joint EDBT/ICDT 2013 Workshops:323–324. https://doi.org/10.1145/2457317.2457373
16. Dadgari D, Stuerzlinger W (2010) Novel user interfaces for Diagram versioning and differencing. British HCI, British HCI
17. Discord—A New Way to Chat with Friends & Communities. (n.d.). Discord. Retrieved June 23, 2020, from https://discord.com/channels/@me/517739346602754076
18. Doboš J, Fan C, Friston S, Wong C (2018) Screen space 3D diff: a fast and reliable method for real-time 3D differencing on the web. Proceed 23rd Int ACM Conf 3D Web Technol:1–9. https://doi.org/10.1145/3208806.3208809
19. Doboš, J., Sons, K., Rubinstein, D., Slusallek, P., & Steed, A. (2013). XML3DRepo: a REST API for version controlled 3D assets on the web. Proceed 18th Int Conf 3D web Technol, 47–55. https://doi.org/10.1145/2466533.2466537
20. Doboš J, Steed A (2012a) 3D diff: an interactive approach to mesh differencing and conflict resolution. SIGGRAPH Asia 2012 Technical Briefs:20:1–20:4. https://doi.org/10.1145/2407746.2407766
21. Doboš J, Steed A (2012b) 3D revision control framework. Proceed 17th Int Conf 3D Web Technol:121–129. https://doi.org/10.1145/2338714.2338736
22. Dourish P, Bellotti V (1992) Awareness and coordination in shared workspaces. Proceed 1992 ACM Conf Comput-Supported Cooperative Work:107–114. https://doi.org/10.1145/143457.143468
23. Fan H, Sun C, Shen H (2012) ATCoPE: any-time collaborative programming environment for seamless integration of real-time and non-real-time teamwork in software development. Proceed 17th ACM Int Conf Support Group Work:107–116. https://doi.org/10.1145/2389176.2389194
24. Field A, Miles J, Field Z (2012) *Discovering statistics using R* (1 edition). SAGE Publications Ltd.
25. *Firebase*. (n.d.). Firebase. Retrieved October 20, 2020, from https://firebase.google.com/
26. *Forums—GameDev.net*. (n.d.). Retrieved October 25, 2020, from https://www.gamedev.net/forums/
27. Galy E, Paxion J, Berthelon C (2018) Measuring mental workload with the NASA-TLX needs to examine each dimension rather than relying on the global score: an example with driving. Ergonomics 61(4):517–527. https://doi.org/10.1080/00140139.2017.1369583
28. *Git*. (n.d.). Retrieved December 22, 2018, from https://git-scm.com/
29. Gleicher M, Albers D, Walker R, Jusufi I, Hansen CD, Roberts JC (2011) Visual comparison for information visualization. Inf Vis 10(4):289–309. https://doi.org/10.1177/1473871611416549
30. Goldman M, Greg D Little G, & Robert C, Miller, R (n.d.). *Real-time collaborative coding in a web IDE | Proceedings of the 24th annual ACM symposium on User interface software and technology*. Retrieved September 2, 2020, from https://doi.org/10.1145/2047196.2047215
31. Google Docs. (n.d.). Retrieved October 24, 2020, from https://docs.google.com/document/u/0/
32. Google Meet. (n.d.). Retrieved October 27, 2020, from https://meet.google.com/
33. Graham M, Kennedy J (2010) A survey of multiple tree visualisation. Inf Vis 9(4):235–252. https://doi.org/10.1057/ivs.2009.29
34. Guerra-Gómez JA, Buck-coleman A, Pack ML, Plaisant C, Shneiderman B (2013a) TreeVersity: interactive visualizations for comparing hierarchical datasets. Trans Res Record (TRR), J Trans Res Board *(2013)*, 21. https://doi.org/10.3141/2392-06
35. Guerra-Gómez JA, Buck-coleman A, Plaisant C, Shneiderman B (2012) TreeVersity: visualizing hierarchal data for value with topology changes. Proceed Digital Res Soc 2012(2):640–653
36. Guerra-Gómez JA, Buck-Coleman A, Plaisant C, Shneiderman B (2011) TreeVersity: comparing tree structures by topology and node's attributes differences. IEEE Conf Visual Analytics Sci Technol (VAST) 2011:275–276. https://doi.org/10.1109/VAST.2011.6102471
37. Guerra-Gómez JA, Pack ML, Plaisant C, Shneiderman B (2013b) Visualizing change over time using dynamic hierarchies: TreeVersity2 and the StemView. IEEE Trans Vis Comput Graph 19(12):2566–2575. https://doi.org/10.1109/TVCG.2013.231
38. Hart SG, Staveland LE (1988) Development of NASA-TLX (task load index): results of empirical and theoretical research. In: Hancock PA, Meshkati N (eds) *Advances in psychology* (Vol. 52, pp. 139–183). North-Holland. https://doi.org/10.1016/S0166-4115(08)62386-9
39. Hegde R, Dewan P (2008) Connecting programming environments to support ad-hoc collaboration. In: 2008 23rd IEEE/ACM international conference on automated software engineering, pp 178–187. https://doi.org/10.1109/ASE.2008.28
40. Herman I, Melançon G, de Ruiter MM, Delest M (1999) Latour—a tree visualisation system. In: Kratochvíyl J (ed) *Graph drawing*. Springer, pp 392–399. https://doi.org/10.1007/3-540-46648-7_40
41. Hoek AVD (2004) Continuous coordination: a new paradigm for collaborative software engineering tools. ICSE 2004. https://doi.org/10.1049/IC:20040207

42. Java | Oracle. (n.d.). Retrieved October 28, 2020, from https://www.java.com/en/
43. Johansson A, Dell'Acqua P (2012) Comparing behavior trees and emotional behavior networks for NPCs. In: 2012 17th international conference on computer games (CGAMES), pp 253–260. https://doi.org/10.1109/CGames.2012.6314584
44. Kodu—Visual Programming Language for creating Xbox games. (n.d.). *Microsoft Research*. Retrieved October 28, 2020, from https://www.microsoft.com/en-us/research/project/kodu/
45. Kolovos DS, Di Ruscio D, Pierantonio A, Paige RF (2009) Different models for model matching: an analysis of approaches to support model differencing. *Proceedings of the 2009 ICSE workshop on comparison and versioning of software models*, 1–6. https://doi.org/10.1109/CVSM.2009.5071714
46. Marcotte R, Hamilton HJ (2017) Behavior trees for modelling artificial intelligence in games: a tutorial. Comput Games J 6(3):171–184. https://doi.org/10.1007/s40869-017-0040-9
47. Microsoft OneDrive | Sign In or Sign up | Free Cloud Storage. (n.d.). Retrieved October 24, 2020, from https://www.microsoft.com/en-ca/microsoft-365/onedrive/online-cloud-storage
48. Moss R (n.d.) *7 examples of game AI that every developer should study*. Retrieved October 16, 2020, from /view/news/269634/7_examples_of_game_AI_that_every_developer_should_study.Php.
49. Munzner T, Guimbretière F, Tasiran S, Zhang L, Zhou Y (2003) TreeJuxtaposer: scalable tree comparison using focus+context with guaranteed visibility. SIGGRAPH 2003 22(3):453–462. https://doi.org/10.1145/882262.882291
50. Namata GM, Staats B, Getoor L, Shneiderman B (2007) A dual-view approach to interactive network visualization. CIKM 2007:939–942. https://doi.org/10.1145/1321440.1321580
51. NodeCanvas—Asset Store. (n.d.). Retrieved July 8, 2019, from https://assetstore.unity.com/packages/tools/visual-scripting/nodecanvas-14914
52. NodeCanvas—Behaviour Trees and State Machines for Unity Game Engine. (n.d.). Retrieved April 5, 2020, from https://nodecanvas.paradoxnotion.com/
53. Nosek JT (1998) The case for collaborative programming. Commun ACM 41(3):105–108. https://doi.org/10.1145/272287.272333
54. Online Collaboration Tools for Modern Teams | Nulab. (n.d.). Retrieved October 29, 2020, from https://nulab.com/
55. Online Diagram and Flowchart Software | Cacoo. (n.d.). Retrieved October 24, 2020, from https://cacoo.com/
56. Online Project Management Software & Tools | Zoho Projects. (n.d.). Retrieved October 27, 2020, from https://www.zoho.com/projects/
57. Pettersson I, Lachner F, Frison A-K, Riener A, Butz A (2018) A Bermuda triangle? A review of method application and triangulation in user experience evaluation. *Proceedings of the 2018 CHI conference on human factors in computing systems*, 1–16. https://doi.org/10.1145/3173574.3174035
58. Programming and Web Development Help | DreamInCode.net. (n.d.). Retrieved October 25, 2020, from https://www.dreamincode.net/
59. Project Management Software by ClickUp™. (n.d.). Retrieved October 27, 2020, from https://clickup.com/teams/project-management
60. Rasmussen, J. (n.d.). *Are Behavior Trees a Thing of the Past?* Retrieved April 4, 2020, from https://www.gamasutra.com/blogs/JakobRasmussen/20160427/271188/Are_Behavior_Trees_a_Thing_of_the_Past.php
61. Renger M, Kolfschoten GL, de Vreede G-J (2008) Challenges in collaborative modeling: a literature review. In: Dietz JLG, Albani A, Barjis J (eds) *Advances in Enterprise engineering I* (pp. 61–77). Springer. https://doi.org/10.1007/978-3-540-68644-6_5
62. Santoni C, Salvati G, Tibaldo V, Pellacini F (2018) LevelMerge: collaborative game level editing by merging labeled graphs. IEEE Comput Graph Appl 38(4):71–83. https://doi.org/10.1109/MCG.2018.042731660
63. Schmeil A, Eppler M, Gubler M (2009) An experimental comparison of 3D virtual environments and text chat as collaboration tools. Electronic J Knowledge Manag (EJKM) 7(5):637–646
64. Scratch—Imagine, Program, Share. (n.d.). Retrieved October 28, 2020, from https://scratch.mit.edu/
65. Sharp H, Preece J, Rogers Y (2019) Interaction design: beyond human-computer interaction. Wiley
66. Slack. (n.d.). *Welcome to your new HQ*. Slack. Retrieved October 27, 2020, from https://slack.com/intl/en-ca/
67. Soroush Ghorashi, S. & Jensen Carlos. (n.d.). *Jimbo | Proceedings of the 9th International Workshop on Cooperative and Human Aspects of Software Engineering*. Retrieved September 2, 2020, from https://doi.org/10.1145/2897586.2897613
68. Tanimoto SL (1990) VIVA: a visual language for image processing. J Visual Languages Comput 1(2):127–139. https://doi.org/10.1016/S1045-926X(05)80012-6
69. *Team collaboration software | Backlog*. (n.d.). Backlog (English). Retrieved October 27, 2020, from https://backlog.com/lp/team-collaboration/

70. The Total Beginner's Guide to Game AI. (n.d.). GameDev.Net. Retrieved October 16, 2020, from https://gamedev.net/tutorials/programming/artificial-intelligence/the-total-beginners-guide-to-game-ai-r4942
71. TIGSource Forums—Index. (n.d.). Retrieved October 25, 2020, from https://forums.tigsource.com/
72. Twiddla. (n.d.). Retrieved October 24, 2020, from https://www.twiddla.com/
73. Unity Forum. (n.d.). Retrieved October 25, 2020, from https://forum.unity.com/
74. Unity Technologies. (n.d.). *Unity Real-Time Development Platform | 3D, 2D VR & AR Engine*. Retrieved March 5, 2021, from https://unity.com/
75. Unreal Engine. (n.d.). Retrieved August 4, 2020, from https://www.unrealengine.com/
76. Valsamakis Y, Savidis A, Agapakis E, Katsarakis A (2020) Collaborative visual programming workspace for Blockly. IEEE Symposium Visual Languages Human-Centric Comput (VL/HCC) 2020:1–6. https://doi.org/10.1109/VL/HCC50065.2020.9127253
77. Video Conferencing, Web Conferencing, Webinars, screen sharing—zoom. (n.d.). Retrieved October 27, 2020, from https://zoom.us/
78. Zacharis NZ (2011) Measuring the effects of virtual pair programming in an introductory programming Java course. IEEE Trans Educ 54(1):168–170. https://doi.org/10.1109/TE.2010.2048328
79. Zaman L, Kalra A, Stuerzlinger W (2011) The effect of animation, dual view, difference layers, and relative re-layout in hierarchical diagram differencing. Graphics Interface 2011:183–190 http://portal.acm.org/citation.cfm?id=1992917.1992947
80. Zaman L, Stuerzlinger W, Neugebauer C (2017) MACE: a new Interface for comparing and editing of multiple alternative documents for generative design. *Proceedings of the 2017 ACM symposium on document engineering*, 67–76. https://doi.org/10.1145/3103010.3103013
81. Zaman L, Stuerzlinger W, Neugebauer C, Woodbury R, Elkhaldi M, Shireen N, Terry M (2015) GEM-NI: a system for creating and managing alternatives in generative design. *Proceedings of the 33rd annual ACM conference on human factors in computing systems*, 1201–1210. https://doi.org/10.1145/2702123.2702398