# ”**A**li**C**e-**V**i**L**lag**E**”
# Alice as a Collaborative Virtual Learning Environment

Ahmad Al-Jarrah and Enrico Pontelli
Department of Computer Science
New Mexico State University
Las Cruces, New Mexico, USA
Email: {jarrah,epontell}@nmsu.edu

*Abstract*—**There is a growing literature demonstrating the importance of collaboration and teamwork in the process of learning computational thinking and the foundational aspects of computer science. While *Collaborative Virtual Environments* are becoming widespread in the software development professional domain and in various areas of advanced learning, their use in the introductory stages of learning computing is still very limited. On the other hand, in recent years, there has been a blooming of attractive programming environments specifically designed to expose young students (e.g., middle school age) to computational thinking. Alice is a very popular introductory programming environment, focused on programming through animations and story-telling. This paper introduces a novel extension of the Alice framework that enables interaction and collaboration among students in the development of programs. In particular, the new version of Alice described in this paper provides support for *virtual pair programming*. The modified version of Alice allows two students to remotely share a virtual world, and interact in its construction; the system supports roles assignments, to allow teachers to control activities and contributions of the two students in the creation of a programming project.**

## I. Introduction

### A. Motivation

The Computer Science community is facing a complex question [1]—how to engage more students in gaining competency in computing (and possibly pursue academic degrees in the field of computing), in order to meet the pressing demand for trained computing workforce. Unfortunately, while more and more jobs demand sophisticated computing skills, the number of students considering careers in this field remains abysmally low [2]–[4]. According to the United States Bureau of Labor Statistics, the number of professional positions in computer science is increasing at a rapid pace. The projections indicate growths in number of positions of more than 20% by 2022 in all core areas of computing, making computing the fastest growing cluster among all occupations. The projected growth in job opportunities is, unfortunately, not satisfied by the current production of academic degrees. This situation is further exacerbated by the severe unbalance in gender and ethnic representation in the discipline. According to statistics from the *Computing Research Association (CRA),* women represent 51% of the US population while only 12.9% of undergraduate degrees in computer science are granted to women [5].

The somber numbers of students pursuing computing can be attributed to a variety of factors—e.g., negative image [6], [7], fear of offshoring [8]. But a critical factor lies in the CS gaps in the K-12 system. While other countries have enhanced their educational efforts in computing (e.g., China has integrated CS throughout the high school math curricula [9]), the CS Teachers Association (CSTA) reports a neglect of CS at the high school level in the U.S. From 2005 to 2009, the percentage of schools offering CS courses dropped from 40% to 27%, due to lack of interest and lack of teachers [10]; CS courses often count only as general electives, not as college-preparatory electives [11]; many of them are more about use of applications than actual CT [12]. Indeed, the literature has repeatedly emphasized the importance of early exposure and training in computing, especially to engage students from traditionally underrepresented groups (e.g., [11], [13]).

The research focus on early exposure to computing has resulted in a number of notable achievements. Two aspects that are of central interest to this project are: **(1)** The development of a number of highly visual and interactive programming environments, specifically designed to serve a student population ranging from the early grades to high school (e.g., Alice [14], Scratch [15]); **(2)** The identification of a number of core issues in the development of competency in computational thinking, that informed the design of novel pedagogical practices and K-12 curricula (e.g., the College Board CSPrinciples draft curriculum [16]). In particular, an aspect that has been recognized both in the various K-12 computing curricula (as a fundamental learning objective) and in several computing pedagogical practices is the central role of collaboration, teamwork and communication.

These two aspects provide the target of opportunity for the research described in this paper. While effective in their individuality, the most popular introductory programming environments are predominantly single-user, with scarce or no support for collaboration and interaction among students *during* the development of applications. In turn, simple (e.g., pair-programming) and complex (e.g., Affinity Research Groups) collaborative computing pedagogical methodologies are poorly supported by programming environments and, to the best of our knowledge, are infeasible in a remote setting (e.g., students working in their own homes, students engaged in online courses, students who are homebound due to disabilities). Our goal is to address this situation.

## B. Summary of Contributions

In this paper, we describe a novel collaborative environment for *Alice programmers.* Originally, Alice has been proposed as an interactive programming environment, freely available, and designed to teach programming (including relatively advanced features, like object oriented programming) to students at the middle and high school level. We propose the design and implementation of new features and functionalities in Alice to facilitate concurrent programming of 3D worlds by a *group of programmers.* We refer to the resulting environment as "AliCe-ViLlagE." *AliCe-ViLlagE* is an application to collaborate and learn in a collaborative virtual environment. By building this environment, we aim to combine the benefits from using Alice in teaching programming and the benefits from using collaborative models in programming and learning, e.g., increased technical experiences and skills exchange, sharing of information, better program quality with fewer bugs, and enhanced student confidence.

The name of this new project, *AliCe-ViLlagE*, captures the two perspectives of this project. First, the name borrows its vision from the view of the internet as a *"Global village"*—used to describe the internet as a community where people work and interact regardless of physical distances; in the same spirit, *AliCe-ViLlagE* allows young students to feel as if they are working together inside a room and on the same machine, while they may be physically located at a distance. Secondly, the capitalization of the letters *A C V L E* in *AliCe-ViLlagE* captures the essence of the new programming environment: **A**lice as a **C**ollaborative **V**irtual **L**earning **E**nvironment (see also Figure 1).

The design of *AliCe-ViLlagE* is also meant to benefit teachers. The collaborative environment is designed to enforce *roles*, thus allowing teachers to structure student teams to ensure inclusiveness, shared responsibilities and enhancing monitoring and accountability. The support for debugging of *AliCe-ViLlagE* will also allow teachers to replay the history of program development, facilitating the identification of strengths and weaknesses of individual team members.

In this first phase of the *AliCe-ViLlagE* project, we focus on extending Alice to enable *pair programming.* In order to experiment with different interactions, we enable in *AliCe-ViLlagE* different variations of pair programming (including both driver/observer and driver/driver versions). This phase is a first step towards building a comprehensive Alice collaborative virtual learning environment for arbitrary groups of programmers.

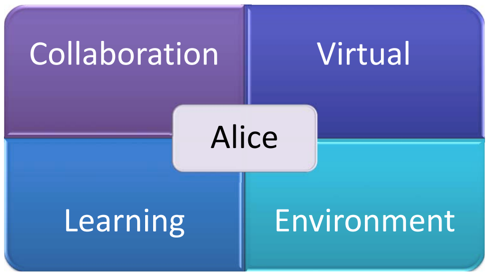


Fig. 1: Alice as a Collaborative Virtual Learning Environment

## II. Brief Literature Review

**Pair Programming:** While the practices of programming have always acknowledged the importance of teamwork and collaboration, it was only in the 1980s that software engineering research started highlighting the benefits of software development activities conducted by teams of two people at the same workstation. Larry Constantine was perhaps one of the first researchers to investigate the benefits of working in pairs in programming tasks, like producing code faster with fewer errors; he referred to this practice as *"Dynamic Duos"* [17]. Working in pairs is not new, but studying pair programming as methodology started by the end of 1990s. The authors of [18] reported that pair programming started gaining traction and popularity around 1995.

Software engineering research has since then widely explored pair programming as the collaboration between two persons working on the same problem (e.g., [19]–[24]). Pair programming is one of the practices of Extreme Programming (XP) and is one of the Agile Methods [21] in software development. In pair programming, two programmers work collaboratively at one computer on the same design, algorithm; researchers emphasized the importance to control such interaction—e.g., the authors of [25] require a single keyboard, monitor, and mouse to be shared between the pair. One of the team members in a pair is assigned the role of the *driver,* where s/he has the control of the keyboard and the mouse; the second member plays the role of the *navigator* (also known as *observer* or *pointer*) [19], [20]. The navigator has a more strategic role, such as looking for errors, thinking about the overall structure of the code, finding necessary information and brainstorming with the driver [21].

Several case studies reported a positive effect of pair programming. The authors of [20] indicate that a good pair programming team is fast, efficient, and effective. According to [20], pair programming leads to fewer bugs, a better understanding of the code structure, higher quality of code, better design and improved morale. The benefits of pair programming have also been observed in the pedagogical context. The investigation of [22] compares performance in programming assignments between two concurrent courses, one taught using the pair programming model and the second without; the results show that students in the pairing course scored higher on programming assignments (M=86%) than those in the non-pairing class assignments (M=67%); the comparison confirmed also that the benefits originate exclusively from the joint programming activities (indeed the two courses provided comparable results on the theoretical materials).

Several studies have been done on pair programming to study the effects of different variables on the programmers' behavior and results. The authors of [23] examine the effect of neuroticism and the final result show that the differences in neuroticism levels did not affect the academic performance for paired students. The authors of [26] study the effect of pair programming on the use of resources and the time spent to switch between them; their results show that pair programming did not introduce noticeable overheads in resource switching; the team members tend to switch less often between tools, thus spending more time in directly productive activities, and they spend more time on a tool before moving to another one.

The authors of [22] conclude that students who work in pairs, produce better programs and pair programming can be used effectively in an introductory programming class. The authors of [24] strongly support such conclusions; they observed that the percentage of students who use pair programming that get a grade C or higher is equal or higher than the percentage of students who work individually. The performance on the exams for students who used pair programming tends to be better than the performance of students who work individually—denoting a joint benefit in the learning process (not only in the actual practice of programming). Finally, students have been observed developing a positive attitude towards collaborative programming settings when using pair programming, and developing stronger individual programming skills—thus, working in pairs does not affect the ability to work individually in the future. Also, the conclusion of [26] include that pair programming affects people's habits, helping them to focus more on productive activities.

**Collaborative Environments:** Pair programming represent the first step towards the creation of collaborative programming (and learning) environments. Collaborative learning activities have been recognized as increasing technical ability; the collaborative aspect allows students to explore learning materials with the backing of a social infrastructure, which creates a "safer" and less threatening learning environment. At the same time, the collaborative aspect introduces a level of accountability and supports exploration of the learning materials from different perspectives and interpretations. Collaborative learning has been shown to be particularly important to engage traditionally underrepresented groups (e.g., [13], [27], [28]).

The author of [29] compared a cooperative learning approach in CS1 to a traditional lecture-based approach. The study has been conducted with 71 students, 34 students in an experimental group (using cooperative learning) and 37 students in the comparison group. The result shows that the scores in the final exam for students in the experimental group are higher than students in the comparison group.

Collaborative virtual environments enable the implementation of collaborative learning without the need of face-to-face in-class interactions; thus, collaborative virtual environments are critical in supporting distance education [30]. In the modern pedagogies, the technology and the implementation of communication are very important to enhance the learning process, especially in the case of implementing constructivism. The final conclusion of [30] include that the enhancement of communication and community of distance education can be met through the use of collaborative virtual environments.

**Alice:** Alice [31] is an interactive 3D programming environment for teaching students programming concepts. Alice was originally developed by a research group at Carnegie Mellon University. Teachers can use Alice as a free educational software to teach computer programming in a 3D graphical environment. Students can build a 3D world to tell a story, build an interactive game, or a 3D animation at the same time as they learn programming. Just as other similar environments (e.g., Scratch, SNAP!), Alice focuses on enabling students to express their creativity in the form of programs, while avoiding the syntactic intricacies of traditional programming languages. The Alice environment is built in Java—and Alice programs can be translated into Java code.

Novice programmers can develop an interactive 3D environment in an easy way; they can add entities, actions/behavior, or events by simply dragging and dropping objects into the world. Alice provides a large number of 3D objects (e.g., people, animals, vehicles) that can be added to the virtual world. Even though Alice programmers do not need to write code a traditional programming language (e.g., Java, C++), the constructs available in Alice are direct abstractions of the typical statements found in an object-oriented programming languages. As such, Alice programmers are exposed to analogous components and methodologies as in traditional programming languages—with the difference that, instead of writing textual instructions, programs are assembled through a graphical process of drag&drop. This method makes programming for beginners faster and easier, removing the complex cycle of editing/compiling/linking; Alice removes the risk of syntax errors and hides several low level details (e.g., memory management). Students can animate the statements, thus gaining a visual representation of the execution of different types of statements. so they will get fast feedback about their statements structure.

Alice has become a very popular tool to introduce young students to programming. For example, the author of [14] used Alice in teaching programming concepts and problem solving concepts to high school students at Ithaca College for two summer sessions. Alice has been the language of choice for several middle and high school outreach programs (e.g., [32]). Researchers have also explored how well Alice prepares students to transition to more advance programming courses and to traditional programming languages. For example, in [33], the authors describe a model to transition students from Alice to Java. Studies conducted over several semesters show a notable improvement in Java exams performance for students that initially explored the core concepts in Alice (with only some weaknesses, e.g., due to the limited exposure to inheritance provided by Alice).

## III. *AliCe-ViLlagE*

In the previous section, we presented Alice as a programming environment and we noticed that the current version of it is not a distributed application. This means that, if a group of programmers wants to work on the same project, they will have to be physically located at the same site and work together on the same machine to build the 3D world.

### A. *Overview*

Programming teams are commonly organized according to an established collaborative software development model—such as pair programming for the case of 2-person teams; these collaborative models establish roles and responsibilities of team members within the development team. The first step to work as a team is to assign the roles for the team members. In general the assigned roles differ according on the implemented model. Typical roles that can be encountered in formal collaborative models include the *drivers* (i.e., a lead programmer, in charge of actively writing code on a machine) and the *navigators* (i.e., in charge of monitoring code development and assisting in debugging); more specialized roles appear in the context of design and brainstorming activities—e.g., roles such as the *time-keeper* and the *presenter*.

In the context of a student team working on an Alice project, the roles are restricted by the naturally single-programmer design underlying the Alice environment. In this case, a single programmer (the driver) will have the control of the mouse and keyboard; thus, only one programmer will be working at any time on the actual code. If the team envisions the presence of multiple drivers, they will require to take turns working on the actual code, exchanging the source code among themselves as each driver terminates his/her tasks.[1] The single file being manipulated will be visible, at any point in time, only by the current active driver. Other programmers in the team will not have access to the latest changes made by the driver until s/he re-sends the updated project files to the rest of the team. The restrictions affect also the other roles of a typical team; for example, the Alice environment forces a navigator to be actually sitting next to the driver, looking at the same screen.

The current implementation of *AliCe-ViLlagE* focuses on two-people teams, supporting the original pair programming model (i.e., a driver and a navigator) as well as allowing variations on this model (e.g., by allowing two concurrent drivers). The *AliCe-ViLlagE* framework is designed to met the following requirements:

- The assignment of roles to team members should be dynamically changeable;
- The system should allow both imposed roles (e.g., a teacher requiring the two students to maintain fixed roles) as well as roles that can be modified on-the-fly by the team-members;
- Each team member should be allowed to view the code on his/her own display, and the two team members should be allowed to collaborate even if not physically at the same location;
- *AliCe-ViLlagE* should provide a synchronous behavior: each change to the code/3D world made by one team member should be immediately reflected on the view provided to the second team member.

The traditional pair programming model requires one team member to serve as the driver (writing code) while the second team member serves as the navigator (reviewing the code being written, debugging, brainstorming). In *AliCe-ViLlagE*, this model of interaction is easily realized with the generalization that the two team members can be interacting at a distance (e.g., each team member is at home). The *AliCe-ViLlagE* enables the interaction through the following mechanisms:

- The synchronous behavior of *AliCe-ViLlagE* allows each modification made by the driver to be immediately visible to the navigator, enabling a consistent view of the 3D-world to both team members;
- The navigator has different mechanisms to provide feedback to the driver—such as a text chat window, a camera/audio live feed, and the ability for the navigator to attach notes/comments to any entity in the Alice code.

[1]This is also causes by the limitations of Alice in supporting modularization and import of code fragments.

The ability to provide dynamic modifications of roles allows *AliCe-ViLlagE* to support more flexible pair programming models, where the two team members can play the two roles (driver role and navigator role) at the same time. There is a significant value in allowing this type of extension:

1) It allows the two team members to alternate in the roles of driver/navigator (e.g., upon switching phase in the project);
2) It allows both team members to act as concurrent navigators—which is important for brainstorming and debugging;
3) It allows both team members to act as concurrent drivers—enabling modular development of an Alice project.

If a team member enters *AliCe-ViLlagE* as a navigator, s/he can extend his/her role to driver at any time and also if s/he is a driver, s/he can change his/her role to navigator. The synchronous nature of the interaction ensures that team members have a consistent view of the code and of the 3D world being developed.

Thanks to these changes, *AliCe-ViLlagE* has become much more than a simple programming environment: it is a collaborative virtual learning environment, promoting collaboration, simultaneous code development, and interaction regardless of physical location. *AliCe-ViLlagE* programmers work on the development of the 3D world by programming in a virtual environment, In other word, the two programmers will collaborate to build the 3D world simultaneously, so that when one programmer performs any transactions (e.g., add an object, create a new method, create a new function, add a parameter for the method, add variables) on his machine, the effect is immediately visible on the screen of the second programmer. Furthermore, consistency controls are present to prevent conflicting concurrent modifications of the same entity.

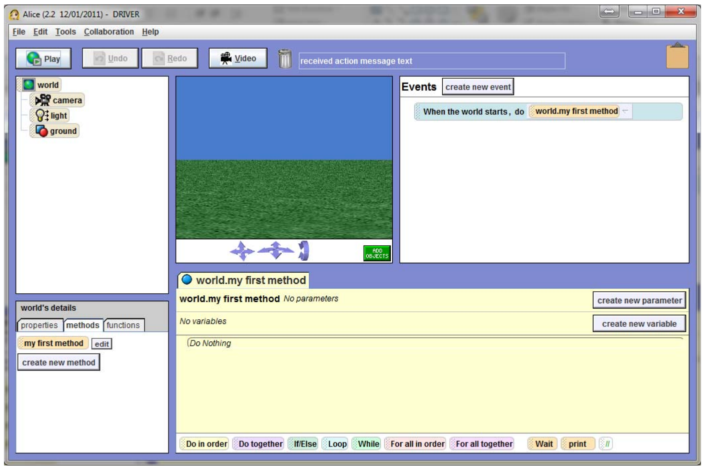


Fig. 2: The GUI for *AliCe-ViLlagE*

At the beginning, the supervisor (e.g., a teacher) assigns the preliminary roles to each team member. *AliCe-ViLlagE* offers a supervisor interface to perform such task. In a typical session, the instructor may start by allowing both team members to serve as navigators to review and discuss the problem specification and the assigned requirements. Once the requirement analysis phase is completed, the instructor may assign one of the team members to serve as driver for the initial development phase. At any point in time, the instructor may force an exchange of roles, to allow the driver to become the navigator and vice versa. In order to facilitate these types of progressions, we are currently designing a component of the supervisor interface that allows the teacher to pre-define the phases of the project and the role assignments for each phase.

### B. Implementation

*1) GUI of* AliCe-ViLlagE*:* The GUI for *AliCe-ViLlagE* (figure 2) is the same as the GUI for Alice with some small changes. We explicitly emphasized the importance of making the novel features of *AliCe-ViLlagE* as nonintrusive as possible, to reduce as much as possible the learning curve of *AliCe-ViLlagE* for students that are already familiar with Alice. In particular, all of the code development tasks of Alice are available unchanged in *AliCe-ViLlagE* with exactly the same visual interface (e.g., add an object to the world by selecting the *"add object"* button and choosing the object from gallery, drag and drop statements into the methods, etc.).

The new components in the *AliCe-ViLlagE* interface are related to the added functionalities of *AliCe-ViLlagE* as a virtual collaborative environment. When the user starts the application, *AliCe-ViLlagE* will prompt the user with a login dialog, requesting a username and a password. These two pieces of information have a dual role; on one hand, they provide security to the users (e.g., protecting the coursework that a student is developing from unintended accesses); on the other hand, the username allows the identification of the user, necessary to place the user in the appropriate pair-team and assigning him/her the appropriate role. Information about users, teams and roles are maintained in an internal database within *AliCe-ViLlagE*.

In the main window of *AliCe-ViLlagE*, the user is presented with the same interface as Alice, with the three new components (figure 4). The first component (figure 4(A)) is a new menu item in the menu bar, called *Collaboration*. The second component is a new button labeled *Video* (figure 4(B)). The third component is a text message field located in the tool bar (figure 4(C)). The Collaboration item (figure 5) is a menu with four options:

1) *"Video Call:"* this option establishes a video conference (synchronized audio and video) with the other team member; the student can also request the video call using the shortcut button (figure 4(B));
2) *"Partner Information:"* this option allows the user to retrieve basic information about the other team member, including his/her identity, his/her current role, and a list of the most recently performed operations;
3) *"Log File:"* this option provides access to a log file containing all the operations performed by either team member, sorted according to the order of execution and time-stamped;
4) *"User Profile:"* this option allows the user to review his/her own profile and manage the information in it (e.g., modify password); the user profile allows the user to also change his/her role, if the supervisor has granted permission to do so.

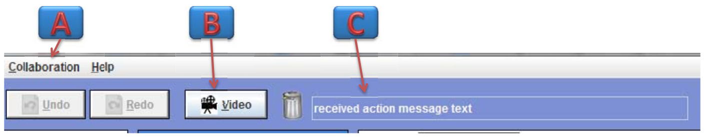


Fig. 4: The three new components in the GUI for *AliCe-ViLlagE*: **(A)** the menu list in the menu bar, **(B)** The Video button, and **(C)** the message bar to show received action

The third component, as in figure 4(C), displays a message, each time a code modification is performed by the other team member—in particular, the message provides a brief description of what change has just been performed (e.g., an object has been added, a method has been modified). This message is important to make the student aware of activities conducted by the partner, since the student might be currently viewing a part of the 3-D world that is unaffected by the modifications being performed remotely. For example, the student might be operating as a driver and working on adding a new object to the world—in this case, the Alice interface is in large mode and the methods panel is hidden. A concurrent modification to a method (e.g., drag and drop of a new statement) performed remotely by a partner will go unnoticed in such scenario, even though it might be of immediate relevance to the student. The message field will prompt the attention of the student on the fact that an exogenous change has occurred (figure 3).

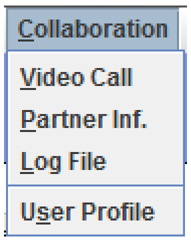


Fig. 5: The collaboration menu list in the menu bar

*2) Users Information: AliCe-ViLlagE* keeps required information about users in an XML database. At the beginning, each user has to login to the system before start working on his/her project (figure 6). The database will have the basic information about each programmer: user name, password, roles, etc. and the basic information about the project that each team is currently working on.

*3) Application-level Communication Mechanisms: AliCe-ViLlagE* is a collaborative virtual environments that provide synchronous view of a shared 3-D world on two different workstations, connected through the Internet. In order to ensure synchronous behavior as well as all forms of communications mentioned earlier (e.g., notifications of changes, audio/video communications), *AliCe-ViLlagE* needs a communication layer that supports the exchange of events among the two remote partners. The transfer of events between users is realized through *RabbitMQ* [34]. RabbitMQ is an open source message broker software, that provides a reliable method to send and receive messages. The Advanced Message Queuing Protocol (AMQP) implemented in RabbitMQ is a suitable protocol needed in our project. The two team members' sessions are connected by two message queues; each team member will

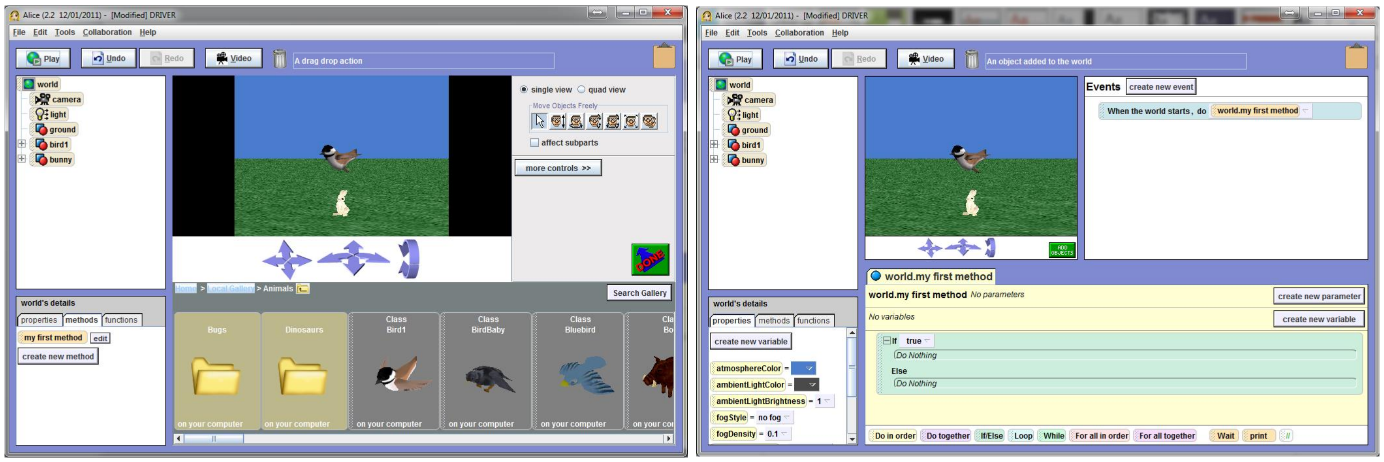


Fig. 3: The GUI interface for the both drivers, (a) the GUI for first driver shows that s/he works on adding objects to the world, where (b) is the GUI for second driver shows that s/he work on drag and drop statements to a method .

Login window
Alice
Learn to Program with Interactive 3D Graphics
a free gift to you from
Carnegie Mellon
Modified By
NM STATE
Computer Science Department
New Mexico State University
login
exit

User ahmad
Password •••

Fig. 6: Login window; view *AliCe-ViLlagE* logo and ask the user to enter his/her name and password.

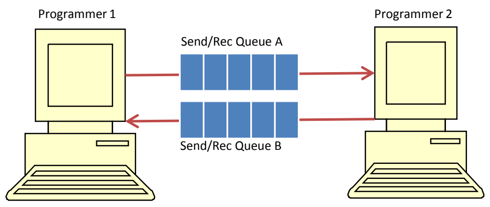


Fig. 7: Send/Receive Queues between team members 1 and 2. Programmer 1 sends a message into queue A and receives a message from queue B

use one queue to send message to the partner and one queue to receive messages from the partner (figure 7).

*4) Consistency:* Even in presence of only two members in a collaborative team, the issue of consistency arises and needs to be addressed. We refer to consistency as the guarantee that the 3-D worlds seen by the two team members are actually *identical* and *non-contradictory*. Identical view ensures that both team members are up-to-date with all changes being made (regardless of who made them). Lack of contradictions implies that the two team member should be somehow prevented from making concurrent and conflicting changes to the same entity of the world. It is easy to imagine that consistency problem may arise at any time while working in *AliCe-ViLlagE*.

Let us further elaborate on the two aspects of consistency. Consistency can be lost as result of two potential situations—resulting in the two team members seeing different versions of the 3-D world. The first situation where a lack of identity in the two 3-D world may occur is when a transaction performed by one team member is not received by the second team member (and, thus, not reflected in the 3-D world see in by the second team member). The second situation relates to the issue of conflicting changes. The two team members (e.g., both acting as drivers) may apply conflicting modifications to the same object of the world (e.g., they both drag and drop a distinct statement in the same method). The order of statements in the method is important, and there is a danger that such order will differ in the two 3-D worlds. For example, figure 8a and figure 8b show the transactions executed by both drivers on each side, resulting in inconsistent methods. Similar problems may occur when both students modify the same property of an entity at the same time—depending on the speed of communication, the two 3-D worlds may result in different values of such property (or, in a less "threatening" case, one of the changes may be lost).

The first situation may occur for various reasons. ranging from shutdown of one of the machines to transient communication problems. The solution for this problem relies on the use of acknowledgment messages. Several well-established schemes have been proposed in the distributed computing literature to address unreliable communication, through the use of delays and acknowledgment messages. The current implementation of *AliCe-ViLlagE* makes use of a relatively simple approach: after each transaction is sent to another machine, an acknowledgement message should be sent back. If no acknowledgement message is received within a given time window, a flag will be set in the log file to show that the transaction has not been properly executed and the two team members are alerted of the problem. In this case, the student may have to wait until the remote 3-D world is properly updated.

The second situation can be resolved using lock messages, generated by the drivers before performing a transaction and associated to the entity which is the subject of the transaction (e.g., driver 1 may generate a lock message associated to method1 before performing a drag and drop of a statement in method1). The actual transaction will be performed only after locking the local entity and receiving an acknowledgment message indicating that the entity has been successfully locked on the remote machine. The locking may fail, e.g., if the entity is already locked by the other team member; in such case, the current entity will be released, the transaction will not occur and the driver will be notified. If the lock is successful, then the transaction will be executed on the current entity and a message sent to the remote machine to request the proper update of the entity. The receipt of the transaction will also correspond to the unlocking of the remote entity. If drivers fail to do a transaction because they noticed that both of them work on the same entity, then they can communicate to coordinate the work. At the end no lock will continue for a while, so no possibility for a deadlock to happen.

*5) Supervisor Functionality:* A teacher will serve in a supervisor role of pair programming teams. *AliCe-ViLlagE* provides a number of features for the supervisor, that include the ability to control the team members from the first step of pairing students to the final step of accessing the final 3-D world to review the work done.

The supervisor has a special interface within *AliCe-ViLlagE*. Through such interface, the supervisor has the ability to manage the students list, and pair students (randomly or according to specific criteria). The supervisor can view the content of all the log files at any specific time, and evaluate the work performed up to that point. The supervisor can step through the operations performed by the students in a team, one by one, progressively recreating the 3-D world developed by the team. The interface allows the supervisor to return feedback to the team, including notes associated to specific entries of the log file (e.g., identifying points where a team member made a mistake). The supervisor interface offers also statistics about each project, broken down by team member (e.g., how many objects each team member created, how many functions were created, etc.).

The supervisor interface allows the teacher to dynamically adjust roles, e.g., as a reflection of the contributions made to the project by each team member. For example, a teacher monitoring the activities in one team may decide to switch the role of drivers and navigators in response to an increasing number of errors made by the current driver. Finally, the supervisor interface provides ability to establish video conferencing and/or text chats with the members of a team.

*6) User Interaction:* As already hinted in the previous sections, *AliCe-ViLlagE* provides members of a team with tools to communicate during the software development process. This is important to allow exchange of ideas, to allow the navigator to offer feedback to the driver, and to support sharing of experiences between team members. Communication in *AliCe-ViLlagE* can occur using two channels. The first channel is audio/video, allowing students to see each others during the software development phase. Video communication allows *AliCe-ViLlagE* to create a connection between team members that is closer to what they experience in traditional pair programming.

The second communication channel is through text messages. Text messages allow fast written exchange of information, enabling the students to attach data references to their messages (e.g., reference to objects or methods in the current code).

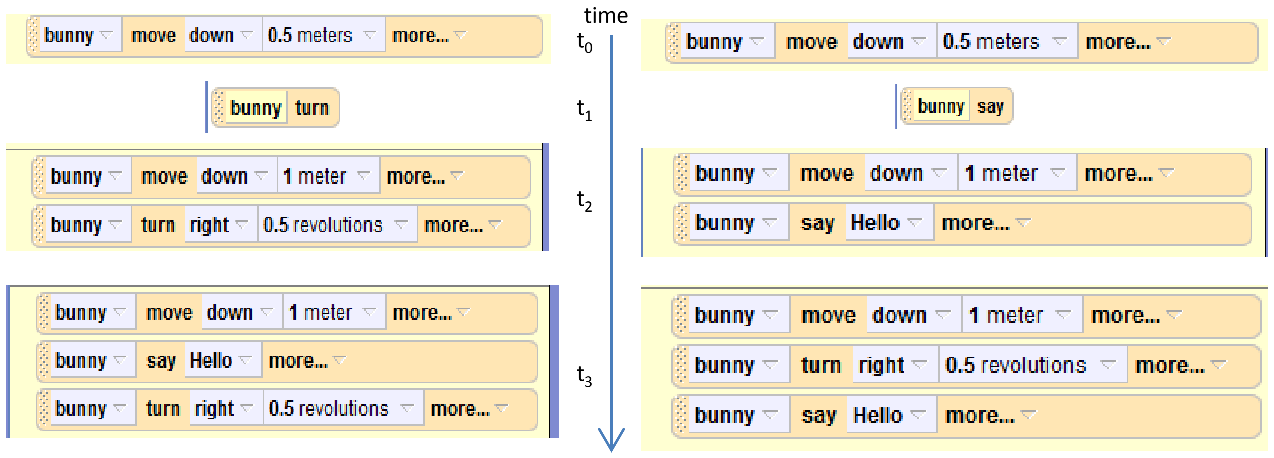
## IV. Conclusion and Future Work

The first prototype of *AliCe-ViLlagE* has been completed and it is now ready for evaluation; it is scheduled to be deployed in a series of high school summer camps during the Summer 2014. We propose to subdivide students in three groups—one group will use traditional Alice and work individually, one group will use traditional Alice and use face-to-face pair programming, while the third group will use *AliCe-ViLlagE*. The membership to the three groups will be rotated during the summer camps. We will conduct focus groups and implement pre- and post-questionnaires to identify strengths and weaknesses of *AliCe-ViLlagE* and inform changes.

A second round of evaluations will be performed in Fall 2014, within the context of an online Computer Animation course based on Alice (CS 117). In this course we will make use of *AliCe-ViLlagE* as an instrument to implement pair programming within an online course, where students do not typically meet face-to-face. We will rely on course grades and questionnaires to compare performance among two groups of students—the first will solve a homework individually using Alice, while the second will rely on pair programming using *AliCe-ViLlagE*. Once again, membership to the two groups will be rotated throughout the semester.

Several extensions of *AliCe-ViLlagE* are currently being considered. First of all, we will enhance *AliCe-ViLlagE* with the ability to support asynchronous interactions, thus enabling team members to work on the project at different times and places; a student logging into *AliCe-ViLlagE* will be able to immediately view the state of the project reached by the other team member, regardless of when the code was written. Similarly, an asynchronous navigator should be able to leave notes and comments about different parts of the Alice project, to be used by the driver next time he/she enters the system.

The other main modification we are working on will allow *AliCe-ViLlagE* to serve teams with more than two members. In this case, we will implement a broader range of roles for the team members, as informed by the existing literature on collaborative software development (e.g., driver, navigator, time keeper, presenter, leader). Just as in the current *AliCe-ViLlagE*, roles will be controlled by the instructor and made dynamic. As part of this extension, we will explore how *AliCe-ViLlagE* can



```
bunny move down 0.5 meters more...

bunny turn

bunny move down 1 meter more...
bunny turn right 0.5 revolutions more...

bunny move down 1 meter more...
bunny say Hello more...
bunny turn right 0.5 revolutions more...
```

(a) Driver I Transaction

```
bunny move down 0.5 meters more...

bunny say

bunny move down 1 meter more...
bunny say Hello more...

bunny move down 1 meter more...
bunny turn right 0.5 revolutions more...
bunny say Hello more...
```

(b) Driver II Transaction

Fig. 8: The transaction executed at both drivers sides according to the time. At time $t_0$, the two drivers have the same state. At time $t_1$, both driver drag a statement to add it to the method, after that the statement added as in time $t_2$, then a transaction message sent to add the statement on the partner side, after receive the message which say to insert the statement at index 1 (after the first statement), the final state on both side as in time $t_3$.

automate the mechanisms underlying complex collaborative models—not just the actual roles, but the dynamic of the roles assignments over the life of the project. As a starting point, we will consider the *Affinity Research Group (ARG)* model [35]. ARG is a collaborative model originally designed to support undergraduate research teams that may include students with diverse backgrounds and levels of competency. ARG has been shown to be very effective to manage small teams, promoting accountability and participation in a face-to-face context. We envision *AliCe-ViLlagE* becoming an automated framework to implement ARG in the context of Alice-based and team-based software projects. This will represent the first ever automated support for ARG, and it will enable to scale ARG from face-to-face interactions to remote interactions.

## REFERENCES

[1] J. Cuny et al., "A Week to Focus on Computer Science Education," National Science Foundation, Predds Release 09-234, 2009.

[2] "Women and Information Technology: By the Numbers," NCWIT, http://www.ncwit.org/pdf/BytheNumbers09.pdf, 2011.

[3] Nat'l Center for Women and IT, "Computing Education and Future Jobs: National, State Congressional District Data," Tech. Rep., 2012.

[4] J. Cuny, "CS 10K Project: Transforming High School Computing for the 21st Century," Presentation, CE21 Community Meeting, 2012.

[5] The Computing Research Association, "Taulbee survey," 2013.

[6] L. Baker et al., *Recruiting Middle School Girls into IT: Data on Girls' Perceptions and Experiences from a Mixed Demographic Group*. MIT Press, 2006.

[7] M. Thom et al., "Understanding the Barriers to Recruiting Women in Engineering and Technology Programs," in *32nd ASEE/IEEE Frontiers in Education Conference*, 2002.

[8] S. Lohr, "Study Plays Down Export of Computer Jobs," *The New York Times*, February 23 2006.

[9] B. Powell, "Five Things the U.S. Can Learn from China," *Time Magazine - World*, no. 12, November 2009.

[10] CSEd Week, "Key Facts About Computer Science," http://www.csedweek.org/key-facts, 2010.

[11] C. Simard et al., "Addressing Core Equity Issues in K-12 Computer Science Education: Identifying Barriers and Sharing Strategies," Anita Borg Institute for Women and Technology, Tech. Rep., 2010.

[12] A. Tucker, "K-12 Computer Science: Aspirations, Realities, and Challenges," in *Teaching Fundamental Concepts of Informatics*. Springer Verlag, 2010, pp. 22–34.

[13] J. Margolis, J. Goode, J. Jellison Holme, and K. Nao, *Stuck in the shallow end: Education, race, and computing*. The MIT Press, 2008.

[14] S. Cooper et al., "Alice: a 3-D Tool for Introductory Programming Concepts," in *Journal of Computing Sciences in Colleges*, vol. 15, no. 5. Consortium for Computing Sciences in Colleges, 2000, pp. 107–116.

[15] D. Malan and H. Leitner, "Scratch for Budding Computer Scientists," in *Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education*. ACM Press, 2007, pp. 223–227.

[16] CS Principles Task Force, "Enduring Understandings, Learning Objectives, Essential Knowledge," The College Board, Tech. Rep., 2013.

[17] L. L. Constantine, *Constantine on Peopleware*. Englewood Cliffs, NJ: Yourdon Press, 1995.

[18] J. Nawrocki and A. Wojciechowski, "Experimental evaluation of pair programming," *European Software Control and Metrics (Escom) (2001)*, pp. 99–101.

[19] E. Arisholm et al., "Evaluating pair programming with respect to system complexity and programmer expertise," *IEEE Trans. Softw. Eng.*, vol. 33, no. 2, pp. 65–86, Feb. 2007.

[20] A. Begel and N. Nagappan, "Pair programming: what's in it for me?" in *Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement*, ser. ESEM '08. New York, NY, USA: ACM, 2008, pp. 120–128.

[21] I. Fronza et al., "An interpretation of the results of the analysis of pair programming during novices integration in a team," in *Proceedings of the International Symposium on Empirical Software Engineering and Measurement*, IEEE Computer Society, 2009, pp. 225–235.

[22] C. McDowell et al., "The effects of pair-programming on performance in an introductory programming course," in *Technical Symposium on Computer Science Education*, ACM, 2002, pp. 38–42.

[23] N. Salleh et al., "The effects of neuroticism on pair programming:

an empirical study in the higher education context," in *Int. Symp. on Empirical Software Engineering and Measurement*, ACM, 2010.

[24] L. Williams et al., "Building pair programming knowledge through a family of experiments," in *International Symposium on Empirical Software Engineering*, IEEE Computer Society, 2003, pp. 143–.

[25] I. Comman et al., "Investigating the Usefulness of Pair-Programming in a Mature Agile Team," in *Int. Conf. on Agile Processes and eXtreme Programming in Software Engineering*, June 10-14, 2008, pp. 127–136.

[26] A. Sillitti et al., "Understanding the impact of pair programming on developers attention: a case study on a large industrial experimentation," in *International Conference on Software Engineering*, . Piscataway, NJ, USA: IEEE Press, 2012, pp. 1094–1101.

[27] G. Crisp and A. Nora, "Overview of Hispanics in Science, Mathematics, Engineering and Technology: K-16 Representation, Preparation, and Participation," HACU, Tech. Rep., 2006.

[28] D. Santiago and S. Brown, "What works for latino students," Excelencia in Education, Tech. Rep., 2004.

[29] L. L. Beck and A. W. Chizhik, "An experimental study of cooperative learning in cs1," *SIGCSE Bull.*, vol. 40, no. 1, pp. 205–209, 2008.

[30] S. Redfern and N. Galway, "Collaborative virtual environments to support communication and community in internet-based distance education," *Journal of IT Education*, vol. 1, no. 1, pp. 201–211, 2002.

[31] R. Pausch et al., "Alice: Rapid prototyping system for virtual reality," *IEEE Computer Graphics and Applications*, 1995.

[32] I. Pivkina et al., "Young women in computing: lessons learned from an educational and outreach program," in *Procs. Technical Symposium on Computer Science Education*. ACM Press, 2009, pp. 509–513.

[33] W. Dann et al., "Mediated transfer: Alice 3 to Java," *Technical Symposium on CS Education*, ACM, 2012, pp. 141–146.

[34] Pivotal, Inc., "RabbitMQ," https://www.rabbitmq.com, 2014.

[35] K. Kephart and E. Villa, "Demonstrating sustainable success: Using ethnographic interviews to document the impact of the affinity research group model," in *Frontiers in Education Conference*, 2008.

[36] A. Q. Gates et al., *The Affinity Research Group Model, Creating and Maintaning Effective Research Teams*, IEEE Computer Society, 2008.