Cheshire: Towards an Alice Based Game Development Tool

Karen Villaverde
New Mexico State University
Computer Science Department
Las Cruces, NM, USA
+1-575-646-1609
kvillave@cs.nmsu.edu

Clint Jeffery
University of Idaho
Computer Science Department
Moscow, ID, USA
+1-208-885-4789
jeffery@cs.uidaho.edu

Inna Pivkina
New Mexico State University
Computer Science Department
Las Cruces, NM, USA
+1-575-646-6237
ipivkina@cs.nmsu.edu

Abstract
This paper presents on-going research that aims to modify the open source 3D programming environment Alice in order to adapt it for use as a development tool for teaching game programming. The advantages and disadvantages of the current version of Alice for game development are described, along with an account of experiences using Alice for game development in the classroom. We set forth the changes that we plan to make to Alice and the challenges that we are facing.

Keywords
Game Development Tool, Alice.

1. Introduction
We have used the 3D programming environment Alice [1] to teach game development in an undergraduate computer animation course for four semesters and in a high school summer workshop course on computer animation for three summers at our university. The objective of the undergraduate course and the course workshop is to learn to program through computer animation and game development and to introduce gently basic computer science concepts. We found that undergraduate and high school students developed very creative and reasonable 3D computer game projects by the end of the course. Also in [2] middle school students in a summer camp built very interesting 3D computer games using Alice.

With the video game industry growing over 40 percent in 2007 to a $17.94 billion business, demand for game developers is strong, but entering the video game industry is a daunting task for a would-be game developer [3]. Many undergraduate and high school students with little or no programming experience aspire to become game developers, but quickly become discouraged when they learn all the knowledge and practice they need in order to create a simple computer game. With Alice, students have a chance to not just learn to program in a fun and interesting way, but also to be able to become game developers of reasonable 3D computer games in one semester or summer.

Even though Alice was not created for game development, it has the potential to be a great 3D game platform if it is modified to add features needed for game development. Our plan is to modify Alice’s “open” source code [1], transforming it into a new game development tool named “Cheshire” that can be used by students with and without programming experience to learn game development.

There are other teaching tools that have been used to develop games that also do not require advanced programming skills. For example: Scratch [4], Greenfoot [5], Game Maker [6], and the Adventure Game Studio [7]. However, these tools do not support 3D game programming. DarkBASIC
[8] and Microsoft Game Studio Express [9] are fine 3D game programming platforms, but they lack Alice’s extraordinary usability advantages for novices.

This paper first gives an overview of the features of Alice that facilitate the creation of simple 3D computer games. Second, we describe our experience using Alice for game development in the classroom. Third, we discuss several drawbacks of using Alice for this purpose. Fourth, we set forth the changes that we plan to make to the Alice source code and the challenges that we are facing.

2. Alice Advantages for Game Development

2.1 User Interface

The user interface in Alice helps students to avoid many syntax errors and frustrations by having a complete drag and drop system (See fig. 1). Students place objects in their virtual world, and then drag commands into user defined methods and functions and into the event editor to animate their objects. This is a big plus for students who are just learning to program since they can concentrate their time and effort on developing their computer game and not learning the complicated syntax and languages nuances of a new programming environment and game engine.

Starting from visual objects in a 3D scene avoids the abstraction which deter many learners of conventional programming languages. Providing all programming constructs and code editing operations via a structured drag-and-drop interface makes syntax errors impossible, while achieving higher usability than the structured program editors developed experimentally in the 1980's [10, 11].

Fig. 1 The Alice drag and drop editor

2.2 Alice Language

Alice includes an interpreter, visual, object-based language that encompasses all the programming concepts that are taught in a first programming course including arrays, functions, passing parameters, and recursion. Alice even introduces students to basic parallel programming concepts.
For example, students can set blocks of code to run either simultaneously or sequentially. This is a great plus since students can acquire valuable experience in parallel programming. Alice’s current implementation is in Java and it is gradually moving towards Java in terms of language syntax and interoperability.

2.3 3D Models and Art Assets
Alice provides hundreds of low polygon 3D models in its built-in class folders. Additional 3D objects can be accessed online through the web gallery or through the Alice Community’s Share Worlds online forum. This is a big plus since it eliminates the need to learn drawing, art, and a complicated 3D modeling tool to create 3D objects. In addition, Alice allows students to import 3D objects from 3D Studio Max and Blender. So students who want to create their own 3D objects have this flexibility as well. Alice also allows students to import 2D images, texture maps, sounds, and music pretty easily.

Re-arranging, resizing, rotating, copying, turning, and tumbling 3D objects in the virtual world is effortlessly achieved by using Alice’s quad view of the virtual world and mouse controls which also allow zooming and scrolling. Furthermore, each of the parts that make a 3D object can be individually manipulated. For example, the arms of a snow man object can be raised, his face turned, etc. This is a big plus since students do not need multiple versions of a 3D model to create an animation. The same 3D model can be used for each frame. Students just need to drag and drop the instructions to move its parts where and how they want them to be.

Alice also allows for objects to be set in different poses. To create a pose, the user makes an object look a certain way by manually moving its parts, and captures the new look. From there, Alice can cycle through these poses and fill in the details between the poses, making animation easy.

2.4 Computer Graphics Handling
Alice provides automatic handling of computer graphics. When objects move in the virtual world, their perspective, lightning, and fog effects are modified automatically. Students can also easily adjust during run time the ambient, directional, point, and spot lights, and fog effects. Therefore, there is no need for computer graphics experience. Since Alice also has its own rendering engine, the need to write code for a specific graphics card is also gone. This is another big plus since it lets students concentrate on the game design and game mechanics instead of studying and coding complicated computer graphics manipulations and dealing with graphics hardware nuances.

2.5 Events and Event Handling
Alice allows for easy event control by the use of its event editor. This is key in setting up game play, as the student can just state a key or button press or condition that causes an event. From here, students can build their game based on which events cause certain actions or animations to happen. The ease of event and event handler creation enables students to create their game play quickly. Depending on the game type being portrayed, movement can range from simple flying to complicated walking movements and hand motions. Setting an event to look for key presses that activate certain animations or move specific objects gives students full control of how objects in their world will move.
2.6 Physics Simulations and Collisions

Normally, in game programming, physics simulations are a formidable task, requiring many lines of code. In Alice, students define the way they want their objects to behave using movement functions. All the movement functions in Alice are defined in meters and can be executed in durations of seconds. This allows the user to create the physics functions of rolling or falling objects. Ambitious students could spend some time creating force, inertia and gravity functions if they like, adding even more realism to their simulation. Collision detection, another important aspect of game programming, is checking whether two or more objects are touching, or near each other. Alice again, has some built-in functions that can be used to program some simple collision detection techniques like the bounding spheres technique. For example, checking the distance between the centers of two objects, and making a decision based on that distance requires just a few mouse clicks. While Alice does not provide them as built-in features, adding simple physics capabilities is an excellent teaching exercise.

2.7 Video Exporting and Sharing

Alice has a video export tool that allows students to make short demonstrations of their games or animations. This allows students to show their games to others who might not have Alice installed on their computer. There is also an option to create a web page with the animation embedded in it. Moreover, Alice has an extremely active online forum where users can post their games or videos and get immediate feedback from thousands of users worldwide.

3. Classroom Experience

Alice has been used to teach a successful computer animation course at our university for four semesters and two high school summer workshops. The course has no mathematics or science prerequisites. The objective of the course is to learn to program with computer animation and gently introduce basic computer science concepts. With Alice all the topics of a first programming course are present as well as basic object orientation and parallel programming. The course covers all chapters of the book Learning to Program with Alice [12]. The authors of this paper are amazed at the design and level of creativity exhibited by the students’ computer game projects. (See Figs. 2, 3, and 4).

According to the games developed by our undergraduate students, and some research on game mechanics in Alice [13, 14], the types of games and game play techniques that work well in the
current version of Alice are: first and third person shooters, shoot ’em up games, adventure games, strategy games, vehicle simulation, interactive movies, and role playing games. A strength of Alice for these games is automatic camera handling. The types of games that have not worked very well in the current version of Alice are: beat ’em up and hack and slash games, fighting games, maze games, 2D games, platform games, music games, sports games, and multi-player games. The main weaknesses of Alice for these games are accurate collision detection, sound handling, and no multi-user support.

Thanks to our success using Alice to develop games, Alice will be used to teach a game design and development course to computer science seniors and graduate students at our university during the spring 2009 semester. It is our belief that the students’ game projects will amaze us even more since these students already know how to program very well and can concentrate all their energy and creativity in the design and development for their games.

4. Alice Drawbacks for Game Development
Even though with Alice students are able to create simple and reasonable 3D games, Alice has important drawbacks. Some general Alice system drawbacks are listed in [15]. In addition to this, we found that for game development the following features are lacking.

- Although the Alice language is an object based language implemented in Java, it lacks several important features of Java. For example, it is not possible to create user defined classes different than normal Alice classes, nor two-dimensional and multidimensional arrays. There is no switch statement and no if statement. Alice has the if-else statement, but writing code with only if-else statements makes code difficult to read sometimes, especially with lots of nested if-else statements. Furthermore, there is no documentation—not even in Alice books—of the functionality of a lot of obscurely named built-in properties, functions, and methods.
- There is no support for dynamic creation and destruction of objects, meaning everything must be placed at load time, with some objects causing long load times for complex scenes.
- Alice does not have a grid to assist in positioning objects; also objects cannot be positioned at a specific location without the help of another object marking the place of that specific location. This makes the code for placing objects at specific locations somewhat cumbersome and difficult. It is actually very difficult to place two objects in the same plane but in different locations.
- The event editor does not allow us to enable and disable events during run time. All the events that will be used in the game must be placed in the event editor at load time.
- Collision detection is not built-in. It must be programmed. Usually, students program simple versions of the common bounding spheres collision detection technique which do not work well all the time.
- There is no physics engine. All physics simulations like gravity and inertia must be programmed.
- It is not possible to have two Alice files open and copy/paste code between them. The only way to copy code from one Alice file to another is to save the code in an Alice class file and then import it.
- The mouse wheel cannot be used to scroll over the code window. Moving an instruction from two faraway places has to be done in several steps.
- There is no support for file input/output and sound input.
Alice offers very primitive debugging tools like watch variables or printing text to a special window when the game is running. Students cannot set break points or run code line by line to debug their games. Furthermore, the Alice compiler error explanations generally are not useful.

- There is no support animated gifs. Static gif files can be imported into Alice but animated gif files cause an Alice error and do not work at all.
- In order to run an Alice game, we must have Alice installed, and run the game in Alice. There is no option in Alice to make the game a standalone executable. There is an option to run an animation in a web browser but not an interactive game.

- The only inputs available in Alice are the keyboard and mouse. Not all keys can be programmed, and the mouse wheel cannot be accessed, and there is no support for joysticks or an Xbox controller.
- Audio capabilities are very limited. It is very hard to stop one audio file and start another one. No audio editing is supported. Audio files imported into an Alice game file cannot be exported. So if for some reason we lose the original audio file, we cannot save and edit it from the Alice game file.
- The current version of Alice does not have any networking capability that would allow students to develop networked games.
- Alice can be unstable at times, especially with games with many objects or lots of code. This forces students to remember to save their work quite often because they never know when Alice will crash.
- Game response time in Alice is somewhat slow. For example, when we press a key, the event handler associated with that key takes a little while to execute. When students run their games, they find it frustrating that the event handlers do not execute right away.

5. Changes We Plan to Make

We do not plan to correct all the drawbacks given above since doing so will take a very long time. But we plan to definitely make corrections that will decrease game memory, response time, and system instability. We intend to implement the following features:

- Add the ability to create and destroy objects at run time, decreasing the memory used dramatically.
- Add the ability to place objects at specific locations with a single instruction specifying the coordinates for the placement. We also want to provide grid support when positioning objects. These features will make the code for placing objects straightforward.
- Add the ability to enable and disable events during run time. This will improve game response time.
- Add built-in collision detection. This will improve dramatically game response time since we plan to implement very efficient collision detection techniques.
- Allow students access to the basic functions of a physics engine: force, gravity, inertia, and particle effects. This feature will allow students to create more compelling visual effects in their games.
- Add the ability to use joysticks and an Xbox controller. We believe that this feature will increase the interest of students in developing more challenging games.
- Implement networking and multiuser support, enabling online and multiuser games. This will be exciting for students since most of them play massively multiplayer online games. Both peer-to-peer and client-server modes of world-sharing will be valuable.
6. CHALLENGES

6.1 Main Challenge
Alice is a project which was developed with public resources and pretends to be open source for political reasons, but its developers clearly would prefer it to be closed and wish to prevent its improvement or extension by other scientists or educators. Our main challenge is to understand the Alice source code which is freely available from www.alice.org. The source code consists of 1,361 files with 180,934 lines of code. All the code is written in Java except for 60 C++ files containing 14,002 lines of code. The public distribution has been stripped of its supporting files for compiling; there is not even a readme file, let alone other design or implementation documentation explaining the architecture, structure or logic of the code. Furthermore, the code is virtually not commented at all, or rather, most likely the useful comments were carefully removed for the public distribution.

We have contacted the Alice programmers who wrote the code; they will not provide any help at all. We will have to figure out the code ourselves. Two of the good things about the Alice source code are that almost all the code is written in Java—which we are very familiar with—and that all files, variables, methods, classes, etc. are very well named.

6.2 Nobody Has Changed Alice
In a discussion held in the Alice community forum, it appears that nobody outside the Alice team has been able to build, let alone change the Alice source code. Their web site claims Alice is open source, but if other scientists are not able to verify a working build from the claimed source code distribution, this claim is unproven. The ideas of Alice have not been used to build Alice based tools outside the Alice team. A member of the Alice team built Squeak Alice [16] which is the Squeak version of Alice, basically an implementation of the Alice 3D authoring tool in Squeak [17].

6.3 Alice 3.0
It is expected that in the fall of 2009, Alice 3.0 will be released as well as its source code. We contacted the chief Alice 3.0 programmer and he suggested that we wait until the source code for Alice 3.0 is released before we make our changes since the new version of Alice will have fewer bugs, will be more stable and will have better features. Alice 3.0 will have the characters from the Sims and will outperform the graphics of Alice 2.0. Games made with Alice 3.0 will have extremely superior production values compared to Alice 2.0. Alice 3.0 will also allow students to type their own Java code using the Eclipse platform. However, we are afraid that that code will be even harder to understand. We have been told that no supporting files for compiling, no readme file, or any other documentation explaining the architecture, structure or logic of the code will be made available either. The Alice 3.0 source code will not be commented as well. Nevertheless, many of the current drawbacks of Alice 2.0 may get fixed in Alice 3.0 and that could ease our work in constructing our game development tool. We will need to make a decision whether to use the Alice 2.0 or Alice 3.0 source code.

6.4 Squeak Implementation
In case we are not successful in changing the Alice source code, we have been contemplating the idea of changing one of three very successful Squeak’s open source code multimedia implementations—mentioned below. Squeak is a modern, highly-portable, open source, full-featured implementation of the powerful Smalltalk programming language and environment. Squeak’s virtual
machine is written entirely in Smalltalk making it easy to debug, analyze, and change. Squeak is a great vehicle for a wide range of multimedia and web applications. Squeak has been used very successfully to build three open source multimedia implementations: Scratch[18], Croquet[19], and Cobalt[20]. Scratch is a new programming language that enables the creation of 2D games, animated stories, and interactive art. Croquet is a platform for creating deeply collaborative 2D games, animated stories, and interactive art. Croquet is a platform for creating deeply collaborative 2D games, animated stories, and interactive art. Cobalt is a metaverse browser and construction toolkit for accessing, creating, and publishing hyperlinked multi-user virtual worlds. As mentioned before, Squeak has also been used to build Squeak Alice which is an approximation of Alice built by a member of the Alice team.

7. Conclusions

Learning a traditional programming language like C++ can take students with no previous programming experience years to master. Learning a fully fledged rendering and physics engine could take these students even longer to produce a simple 3D computer game. However with Alice, in a few weeks time, a student with little or no programming experience can develop simple and reasonable 3D computer games ready to show and receive feedback on. In this paper we covered the advantages and disadvantages of using Alice as a game development tool. We listed the features of Alice that we want to modify and those features that we want to add to Alice to create a new game development tool. We also listed the main challenges that we are currently facing.

8. Acknowledgements

This work was supported in part by NSF Grants CNS-0540592 and CNS-0836632.

References