Visualizing Loops and Data Structures in Xylem: The Code of Plants

Heather Logas, Richard Vallejos, Joseph Osborn, Kate Compton, Jim Whitehead
Center for Games and Playable Media
University of California, Santa Cruz
Santa Cruz, CA 95060
{hlogas, rvallejo, jcosborn, kcompton, ejw}@soe.ucsc.edu

***Abstract* – The visual representation of software data structures is especially relevant to the creation of games which crowdsource science problems to a gaming community and to educational games which seek to teach computer science concepts within the context of computer games. Xylem: The Code of Plants is a game designed to crowdsource formal software verification via loop invariant specifications. Due to the nature of this project, it was important that the game 1) appeal to a large audience, 2) support a wide variety of data structures and 3) hide any actual data from the source code that was generating the game levels. To these ends, a method of visualizing data structures was developed that features a consistent plant-based narrative metaphor, is flexible enough to accommodate multiple types of data structures while maintaining narrative integrity, and obscures all real data from the target source code.**

***Index Terms* – Program visualization, data structure visualization, games and software engineering, software verification games, formal verification, loop invariants.**

I. Introduction

Computer games about computer software face an interesting challenge when it comes to representing core program concepts such as variable values and data structures. A simple and straightforward approach is to directly represent them in programming language syntax (such as with *Pex4Fun* [14]), but this lacks the visual interest and excitement that are a fundamental attraction of computer games. More ideally, a game can use a visual representation of these core concepts, thus improving the visual appeal of the game, and permitting a game to have a consistent overall visual theme.

Visual representation of variable values and data structures is a challenging problem. Consider the representation of integer variables. For small values, a value can be represented using repeated images of some item. If a variable has a value of five, the game can depict five flowers on screen. But, what if a variable value is closer to the maximum integer value? Further, lists are pervasive in software and raise even more representational challenges since they need to represent both individual element values, as well as the list as a whole. This implies that the visual theme for the list needs to be consistent with that of individual list elements.

Xylem: The Code of Plants is a game designed as a tool for crowdsourcing formal software verification. More precisely, by playing the game, players find loop invariant specifications. To support this, Xylem presents information to the player about how a specific loop (for loop, while loop) behaves, and it must do this without presenting the surface program text to the player. Xylem accomplished this feat by presenting players with the value held by variables at different points in the execution of the loop. Players examine these variable values, and then write algebraic equations that are true across all loop iterations. These equations are (partial) loop invariant specifications, which help reduce variable value uncertainty when performing formal value analysis of software systems. We have previously published a more detailed paper exploring Xylem's game design and scientific objectives [16]; this work focuses specifically on the visualization, situating our particular approaches and techniques in the context of other program visualization work.

Xylem is intended to support many different types of data structures, so as to provide the game with a wide expressive range. With a broad set of supported data structures, the game can support a wide range of actual program loops in the game, thus increasing the value of the crowdsourcing approach. This created the challenge of designing a set of data structure visualizations that held together with a consistent narrative – that of the game. As a result, the production of Xylem: The Code of Plants afforded the opportunity to explore visualizations for different data structures.

Explorations into different ways of displaying data were driven by the need to design a game that appealed to a large audience, of the sort that might play casual games online. While early prototypes of Xylem were entirely numbers based, profiles of potential audience members showed a discomfort with numbers and math. It therefore became important to develop a screen design that would support the verification work needing to be done while at the same time creating a narrative space that would be appealing to a wide variety of players. The concerns of game design, therefore, indicated the development of a coherent visual metaphor across multiple data structure types.

The key contribution of this work is a consistently themed visual representation for software loops, including integer variable values and lists of the same, whose values

change over multiple iterations of the loops. A statement of goals is provided, listing issues that must be addressed by any game wishing to have a consistently themed visual representation of loops and software data structures.

II. Related Work

Visualizing the structural elements of a program, such as the body of a loop, has traditionally been achieved by the use of both static and dynamic approaches [5].

The static approaches present a visual analogue of the textual source code, usually favoring diagrammatic representations of control structures. In an early paper on formal verification, Floyd uses flowcharts to visualize the control structure of program loops, annotating his diagram with invariants to show the importance of such statements in the proof of a program's correctness [4]. Dynamic approaches present the behavior of a program structure. Algorithm animations are a common technique for presenting the trace of a program. Baecker's classic video "Sorting Out Sorting" has left a lasting motivation for the visualization of algorithmic behavior [7]. By providing a state-by-state trace of algorithm output over time, such algorithm animations support fast comprehension, and cross-comparison of approaches.

Educators have found the need to illustrate the abstract fundamentals of computer science through the use of visualization. Astrachan introduces the idea of a "picture invariant," an informal diagrammatic representation of a loop invariant, advocating its use in teaching students how to use loop invariants early in a CS curriculum, without requiring an understanding of formal logic [10]. The field of algorithm visualization focuses on providing a visual representation of the execution of a wide range of algorithms, and involves visualization of the data structures used in the algorithm [12]. The Algorithm Visualization Portal at wiki.algoviz.org provides a catalog of many algorithm visualizations, including linked list visualizations. These overwhelmingly use a utilitarian visual representation involving boxes and arrows and are inappropriate for use in games since they do not use an interesting visual theme. Some algorithms employ algorithm-specific visualizations, such as showing a bar chart or line to show values progressing from sorted to unsorted states [13]. Such visualizations typically only apply to a limited range of algorithms, and are not useful when attempting to represent a broad range of possible algorithms. An algorithm-specific approach doesn't work when the algorithm isn't known a priori, as is the case with loops in Xylem.

Ginat discusses the use of mathematical games in demonstrating the thought process of invariant construction [11]. More recently, Eagle et al. [8], and Boyce et al. [9], have developed computer games for teaching the mathematical and syntactic fundamentals of CS. In the game *Wu's Castle*, a game developed by Eagle et al. and targeted to first year CS students, players solve puzzles by tuning the parameters of "for" loops to manipulate array data structures. Boyce et al.'s *BeadLoom* invites player to recreate goal images through the use of basic programming. Targeted to middle and high school students, BeadLoom introduces fundamental concepts of graphics and iterative thinking.

In contrast to algorithm animations, where the aim is to show an abstract representation of loop dynamics in order to give a "big picture" of an algorithm's behavior, in Xylem we adopt a "film strip" representation of the iterations of a loop, which we believe allows for both a global perspective of overall trends as well as the detail needed to construct invariants specific to data. As compared to existing work on the visualization of data structures, Xylem applies a universal visual metaphor across multiple types of data structures. Since this metaphor is visually appealing and avoids the classic boxes and arrows visualization approach, it provides a more interesting thematic setting for games about software.

As part of the DARPA Crowd-Sourced Formal Verification program four other projects (in addition to Xylem) were created – Stormbound, Flow Jam [18], Ghost Map, CircuitBot. Stormbound shares similar goals to Xylem, in that it also uses invariant finding of loops as a way to make progress on formal software verification. However, instead of asking players to identify patterns with mathematical equations, Stormbound presents them with a spatial interpretation of the loop data which is represented as icons ("sigils") on a grid. Players can also combine smaller statements into larger ones by selecting a particular pattern and then selecting another. Both Flow Jam and Ghost Map provide graphical representations of data flow and control flow within software, and broadly focus on ensuring a particular condition holds across a particular path. Circuitbot focuses on pointer analysis.

III. Goals

In order to meet goals for the project of which Xylem was a part, there were a number of constraints that needed to be addressed:

1. Create an enjoyable game that would appeal to a large audience, in order to bring as much crowdsourcing power to bear as possible.
2. Give players the tools to, by way of playing the game, contribute to formally verifying the source code from which the game levels are derived. This should be constructed in a flexible manner, so that potentially any provided source code could be converted into game levels and distributed to players for processing through play.
3. Obscure said source code, in order to prevent potentially malicious behavior.

These constraints fed directly into the development of the visualization methods deployed in Xylem. Since the game needed to appeal to the large casual gaming audience, many of whom do not necessarily associate working with math and numbers as the height of entertainment, the need to abstract out or obfuscate the number aspect of the experience was key. At the same time, this audience was not anticipated to be

overly computer science literate, and so there was a need to present domain-specific information in a way that is understandable to users who may lack domain knowledge. Along with this was the need to make interaction with that domain-specific information motivated and clear, without having to introduce domain-specific notation and terminology. And, in keeping with the expectations of casual game players, an approach was needed that would support a consistent fiction within the game narrative.

To further complicate manners, it was an imperative part of the overall project to not allow players direct access to any of the source code of the target software. This includes showing even small snippets of actual source code. Therefore, the visualization must not only show the game data in a pleasing and consistent manner but also must effectively obfuscate the code such that members of the public are not able to reconstruct any part of the software.

The approach chosen to address the problem of formal software verification is to allow players to discover the invariants of loops in the target software. From a visualization standpoint therefore, the need arose to present on screen the inner mechanics of a loop, and further to visualize a wide variety of data structures that might occur within a loop. Since the game needed to be flexible enough to potentially handle any arbitrary data that was handed to it, a major concern of the visualization component of the project was to be able to represent a wide variety of data structures on screen.

Within the fictional context of Xylem, players navigate a newly discovered island by investigating mysterious plant species. As they make discoveries, players gain points and levels, allowing them to unlock new areas to explore. The game is open ended, with no set winning or losing condition. More information about the player experience can be found in previous work by the authors [16].

IV. Towards a Coherent Metaphor

Early prototypes of the game that would become Xylem featured a space-like theme with an abstract UI design and all variables from the loop represented by the numbers being generated by said loop. In an effort to make Xylem more accessible to the identified target audience, it was important to replace those numbers wherever possible with symbols. These symbols needed to be intuitive to use while supporting the software verification work.

A. Complex Plants

Plants are a familiar part of most humans' existence in one way or another, and to many are appealing and friendly. In user-tests, players could easily distinguish between different plant features as representing different variables. Using plant features that change at different stages of a plant's growth made enough sense to players that it eliminated the need to explain too much backstory before they could be productive.

Another advantage to working in the plant kingdom is that it offers the flexibility necessary to represent a wide range of data structures. Because of the origination of the game level data from arbitrary pieces of source software, the data structures which would appear in the game could not be predicted with any certainty. By surveying different varieties of plant forms, a system for potentially visualizing integers, one and two dimensional arrays, linked lists, stacks, queues, links and graphs, as well as a modular visualization concept for undefined data structures was developed.

Early attempts to use plants as a visual metaphor employed many different plant features as symbols for variables. A list of roughly thirty different plant features that could be drawn from as symbols for variables – number of petals on a flower, height of plant, number of stamen, number of flowers on a plant, number of leaves on a plant, number of thorns, etc. – was developed. Eventually game production concerns prevailed, and in order to streamline the creation of potentially thousands of plants, a procedural flower generator was created. In the current version of Xylem, each integer loop is represented by a plant that has one or more types of flowers growing from it, each one representing a single variable. Players have so far seemed unconcerned with the botanical impossibility of such a scheme.

In order to explain moving back and forth through iterations of a loop (which in the game narrative shows different versions of the same plant), some rather convoluted solutions were attempted. One concept involved the narrative conceit that the player was adding some sort of serum to the plant in question which made it grow and shrink. Ultimately, this was simplified by including in the game fiction the notion that the player is examining different "growth phases" of the plant, implying that the player is looking at slides containing specimens of the same plant at different stages of maturity.

V. Data Structures Implemented in Xylem

In Xylem: The Code of Plants, the player takes on the role of a field botanist in the 1920's, working with others to build a plant taxonomy for the unusual flora endemic to the island of Miraflora. While in the field, the player is equipped with a Flora Phase Comparator (FloCom) which is an all-in-one workstation for botanists.

A. How the FloCom Interface Conveys Loop Data

Players use the FloCom to compare a plant species in different phases of growth and to note salient patterns of inflorescence using mathematical symbols. Within the narrative of the game, each plant species can be described succinctly by a unique mathematical signature, which in game is called an "observation." Players collaborate to rank observations in terms of how well they characterize the inflorescence pattern across growth phases for a particular plant species.

The interface of the FloCom is divided into two main sections. The top half displays a sequence of plant sheets, meant to suggest the dried, pressed flower specimens which are mounted to the pages of herbaria [2]. Each plant sheet

(a) Hypothetical "for" loop

```
a = 2; b = 2/3; c = 2; d = 2;

for (i = 1; i<= 4; i++){
   a = a * c;
   d = a;
   c = d;
   b = b * 2^(2^i);
}
```

(b) Values of loop variables after each iteration

i (#)	1	2	3	4
a	4	16	256	65536
b	6	24	384	98304
c	4	16	256	65536
d	4	16	256	65536

(c) The main gameplay screen (FloCom)

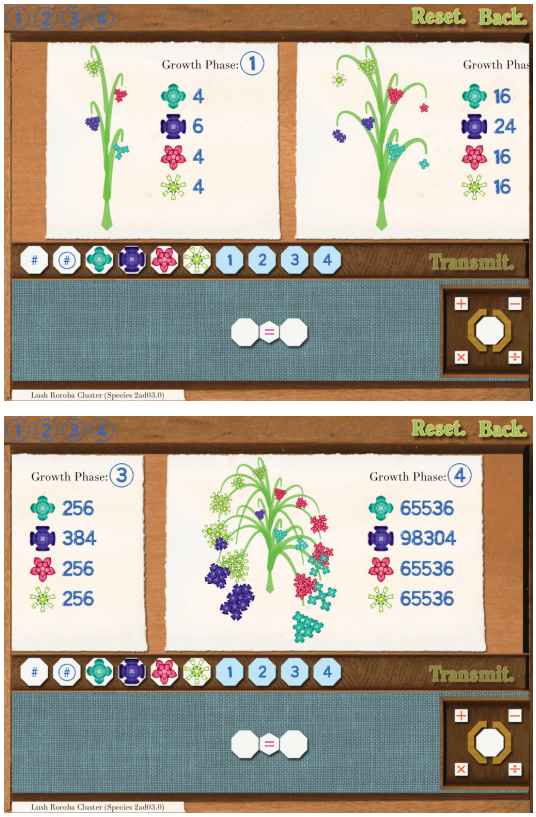


Fig. 1 How Xylem conveys loop data. (a) Hypothetical "for" loop – the mechanics of a loop would, in fact, be unknown to Xylem and the player. We include it here to facilitate the example. (b) Trace of "for" loop, showing the mapping between loop variable to "flower variables." (c) In game representation of loop data.

displays a specimen at a particular phase of a plant species' growth. The sequence of plant sheets are imagined to demonstrate the raw physical data, such as would be collected by a field botanist. Interpreting that data is the task of the player. The bottom half of the FloCom interface is dedicated to providing the player a workspace to describe the sequence of plant sheets using a basic mathematical vocabulary. To express observations, the player uses a tile set, similar to *Scrabble* tiles or magnetic poetry, but with mathematical symbols rather than letters or words. Expressions are developed using the common infix notation system, where operators are placed in between two arguments and operations are ordered in the conventional binding order for functions [3].

The FloCom interface allows a player to build loop invariants while getting dynamic feedback about the appropriateness of the player's input. The validity or partial validity of a player generated observation is expressed via highlighting of the entire plant sheet. If the player's observation describes a slide accurately, the sheet is highlighted green; otherwise, the sheet is highlighted red. In the case when the values compute to numbers outside the allowable range (Xylem allows only integers, zero or greater but less than or equal to 99,999), the plant sheet is highlighted yellow.

B. Visualizing Integer Variables and Values

The flowers on each specimen represent quantities through the use of a positional notation system, where the lesser orders of magnitude are represented nearer to the main stem of the plant.

Fig. 2b provides an example of how two values are represented in game. Here the values for two variables are expressed at their fourth iteration: the red flower variable evaluating to 32,768 and the cyan flower variable evaluating to 256. The values are visualized by successive stems supporting floral clusters; these stems are known as peduncles. The longer and more weighted-down a peduncle, the greater order of magnitude it represents.

C. Visualizing Array Variables and Values

For loops that perform operations on array data structures, the botanical metaphor is further extended by introducing the conceit of a root system. In such problems, the plant page is split into two regions separated by a "soil-level line": above the line, the flowering peduncles convey the values for each variable while below, a set of lateral roots conveys the state of an array at a given loop iteration. The index variable of an array is represented here as distances from the tap root along a lateral root. Within the context of the narrative, on the lower half of the plant sheet the root system is mounted

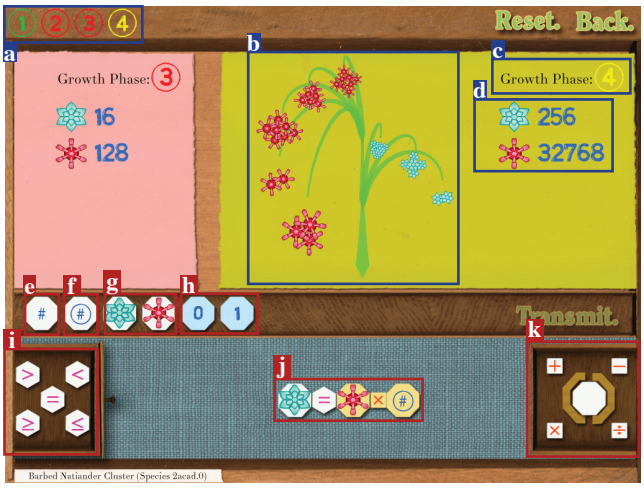


Fig. 2. The Flora Phase Comparator. The upper half, Plant Sheets: (a) growth Phase indicator lights – green: a valid observation; red: an invalid observation; yellow: an observation that evaluates out of bounds, (b) a rendered plant (c) growth phase number, highlighted yellow to indicate a value outside the bounds, (d) flower-variables and their corresponding values. The lower half, the Observation Bed – (e) number input tile, (f) index variable tile, (g) flower variable tiles, (h) bonus tiles – represent the initial values of the loop variables, (i) relational tiles, (j) operator tiles.

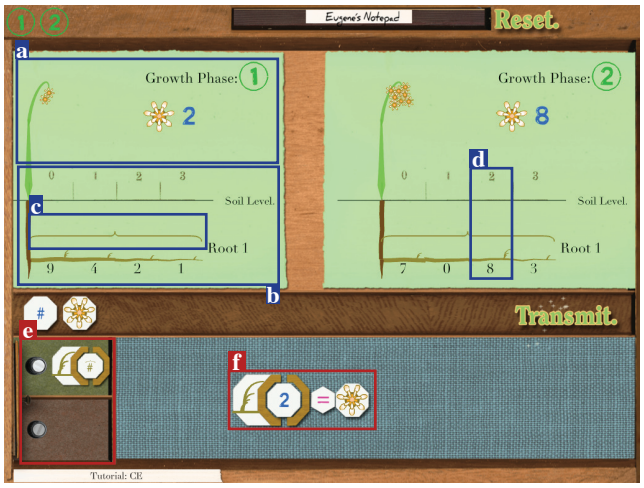


Fig. 3. Visualizing Array Variables and Values. (a) The top-half of the Plant Sheet as described in Fig. 2. (b) The root system, displays array values and indices. (c) The curly bracket object, visually shows the range to be considered by the array variable. (d) An array value and its index. (e) The array variable panel, supports up to two arrays. (f) By default the array variable references all values within a specified range. In this example, the array variable includes only the 2nd index. So, here the invariant can be read: "For all iterations, the value at the 2nd index of the array Root 1 is equal to the yellow flower variable."

to allow the player to perform a line-intercept sampling [1], whereby root hairs are sampled if a line segment, called a "transect," intersects the root hair. The transects are enumerated on the ruler graphic, whose labels occur at inch intervals and correspond directly to the index variable of the array. At each transect, a lateral root will display the number of root hairs on that root at that transect – these numbers coincide with the array values.

For many algorithms a useful loop invariant might only describe a portion of an array – such is the case with many sorting algorithms, where one might want to specify that the entries to the left of a pointer variable are "sorted." In order to allow players a way to specify the values within a range of indices, a curly bracket object is included to accompany each lateral root. The curly brackets visually show which array indices are referenced in terms of the player's observation. Array puzzles also include additional tiles and game panels to allow a player a means to symbolically describe a ranges of indices. See Fig. 4 for a solved example.

To tie this to the calculus of invariants: each array variable is paired with a single universally quantified integer variable which ranges over the length of the array; the invariant applies if this integer is within the array's bounds. By pressing a button on their FloCom, players can slide in a distinct equation-defining board with this quantified variable in the center and lower and upper bounds on the left and right sides. In this way, players define the sub-array for which their invariant holds. Our interface is currently restricted to contiguous ranges, but one can imagine the inclusion of tiles yielding Boolean truth values to express more complex quantification.

VI. Special Challenges

While deciding on the plant metaphor, two representational challenges emerged.

A. Negative Numbers

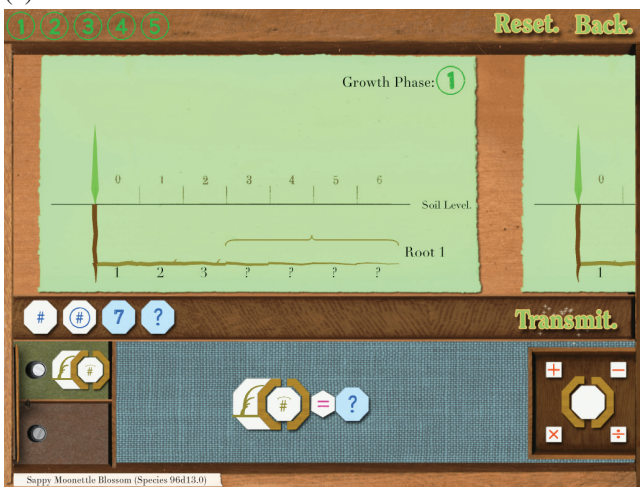
Although negative numbers are not implemented in the current version of Xylem, multiple solutions to this design problem were developed. One of these solutions depended on representing negative numbers as a damaged version of a plant feature. For example, in a problem with two variables represented as different colored flowers, a negative value would be conveyed with one set of flowers that was brown or wilted. Another visual solution to this potential problem was to show positive numbers as plant features growing from upturned branches of the plant, and negative numbers as features growing from downturned branches (in botany this is known as phototropic and pendulous morphology, respectively).

B. Very Large Numbers

Since the proposal for the Xylem visualization scheme is based on literally representing a number onscreen as distinct plant features (flowers, leaves, etc.), the question arose of how to handle very large numbers. Some loops in the game use numbers in the thousands. How many flowers could be possibly readable to the player at one time?

Since we had already decided to present a number on screen for each variable to make it easier for some players to discern patterns, we decided to take a more abstract approach. We realized that showing a precise number of features was less valuable than giving a player a quick visual indication of whether the numbers were going up or down, and their relation to other variables in the loop. Simply showing a blob of

(a)



(b)

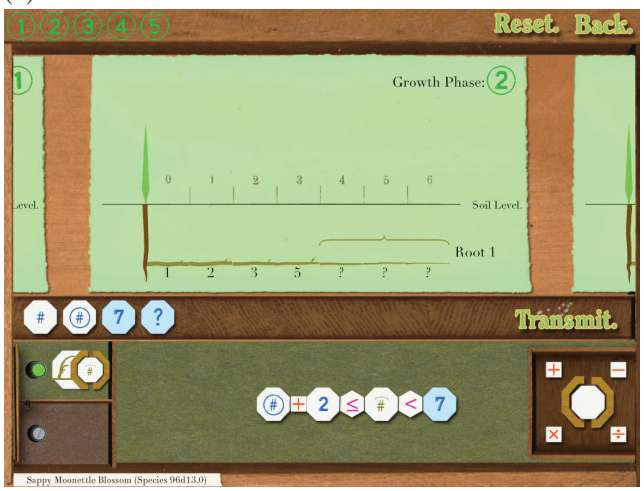


Fig. 4. Specifying dynamic ranges. The array variable here references a range of indices specified in the range editor panel. To edit the array index range, players can toggle the range editor panel. (a) The "observation" here reads: every array entry within a specified range is undefined. (b) The range editor panel is used to constrain the indices. Here the range of indices is related to the loop iteration number, specifically, describing one fewer array entry each iteration.

color with slight delineations when the numbers became too high was suggested, however we still wanted an approach that was consistent. We therefore used a combination of flower position and number to create a sort of flower "code" that could be used to display any number up to 99,999.

VII. Using Plants to Represent Other Data Structures

In the deployed version of Xylem, we were able to represent loops containing integer variables, and integer linked list variables. However, the plant metaphor used in Xylem has substantial representational flexibility, and is capable of handling additional types of data structures, even though they were not implemented. These are described below.

Linked lists, stacks and queues can all be represented in a similar fashion to the linear arrays that are currently in Xylem. The "array length" tool may not be appropriate for these data structures due to the way lists, stacks and queues are more plastic in this dimension than arrays. A new way to describe the length of these structures would therefore have to be investigated. Additional plant features could be added to roots to represent more information about the data contained in the data structure.

One way to visualize a two dimensional array is as a flowering shrub. The array is mapped as a grid over the shrub, and each cell of the array contains a discrete number of plant features such as flowers and leaves which represent data contained at each array index.

Tree data structures can be represented by plant kingdom trees, which have many possible features (leaves, cones, fruit, flowers, etc.) that can be used to represent types of information contained by the data tree.

From a side view, rhizomatous plants appear to be a linear structure. When viewed from the top down however, it becomes clear that actually they grow spread out over a large area. Diagrams of rhizomatous plants from a bird's eye view strongly resemble abstract depictions of graph data structures. The "crowns" of these plants (where the above-ground part of the plant comes through the soil) can have a variety of features that can be used to describe data, such as shoots, leaves, flowers, etc.

VIII. Future Work

In addition to the above standard data structures, a speculative modular design was created to handle any arbitrary data structure that might find its way into the game.

Complex data structures tend to have multiple combinations of primitive data structures, such as a tree where leaves are lists, or lists of lists. The plant metaphor provides two approaches to this problem. First, a berry bush can visualize complex data structures, especially ones where the structure is not known a priori. Data nodes are mapped to berries. Links between nodes are berry stems and vines. These stems and vines do not have to conform to botanical rules; they can be constructed purely based on the needs of the data structure. Currently accessible nodes are represented by ripe berries, whereas nodes not currently accessible are shown as fruit that is not yet ripe. If necessary, different levels of ripeness can be employed to give the player further information.

An alternate approach is to adopt a zooming scheme, where players zoom into and out of different plant features, to uncover data at different levels of resolution. A botanical tree, for example, may grow a flower which in itself represents a node on the data tree containing an array, with vari-

ables represented by number of petals, visible seeds, stamen, etc. This approach requires some foreknowledge of the kinds of data structures involved, so appropriate visualizations can be selected at each layer.

The value of a project like Xylem increases when it is able to represent very complex loops which manipulate complex data structures, as simpler loops can often have their invariants discovered by automatic techniques. As loops become more complex, the ability for automatic systems to solve them goes down, and the utility of having human solvers increases. However, this requires representing a wide range of complex data structures. Xylem, in its current version, is a start at solving this more complex problem, for just simple integer loops, and array/list data structures. Research into more complex data structures such as trees and graphs is needed.

IX. Conclusion

Games that require a faithful depiction of computer software face challenging visual representation issues. Xylem: The Code of Plants needed to visually represent loops, integer variables, and lists of integers, all using a consistent visual metaphor that was appealing to a casual game audience. The approach adopted of using a plant and flower metaphor has sufficient expressive range to represent a wide range of loop examples. Xylem has had over 2,100 downloads to date, a respectable showing for a research game. While these players (who are anonymous due to project constraints) have not been surveyed as to their reaction to the visualization approach in Xylem, there is anecdotal evidence that audiences who are not likely to be pulled into a mathematical game were attracted by the game's botanical theme and enjoyed the experience of making discoveries about virtual flowers [15], (although reviews were not universally positive [17]).

Our core contribution is a visual representation of loops, integers and lists that is narratively coherent, and moves beyond the traditional boxes and arrows visual representation. This visual metaphor holds promise for use in other future software games – both those meant to aid in computer science problems and those meant to teach computer science concepts. Our hope is this will shift thinking about how to visualize data structures and algorithms away from purely utilitarian representations and into more expressive and appealing ones.

References

[1] Kaiser, L, Unbiased Estimation in Line-Intercept Sampling, *Biometrics* 39. pp 965–976. 1983.

[2] Bridson, Diane M., and Leonard Forman. *The Herbarium Handbook*. Kew: Royal Botanic Gardens, 1992.

[3] Baber, Robert, L. *The Language of Mathematics*. John Wiley & Sons, 2011.

[4] Floyd, R. W. Assigning meanings to programs. Mathematical aspects of computer science, 1967.

[5] Diehl, Stephan. *Software visualization: visualizing the structure, behaviour, and evolution of software*. Springer, 2007.

[6] Furia, Carlo A., Bertrand Meyer, and Sergey Velder. "Loop invariants: Analysis, classification, and examples." *ACM Computing Surveys (CSUR)* vol. 46, 3. 2014.

[7] Baecker, Ronald M., with the assistance of David Sherman, "Sorting Out Sorting", 30 minute color sound film, Dynamic Graphics Project, University of Toronto, 1981.

[8] Eagle, Michael, and Tiffany Barnes. "Wu's castle: teaching arrays and loops in a game." *ACM SIGCSE Bulletin*. Vol. 40. No. 3. ACM, 2008.

[9] Boyce, Acey, and Tiffany Barnes. "BeadLoom Game: using game elements to increase motivation and learning." *Proc. of the Fifth Int'l Conference on the Foundations of Digital Games*. 2010.

[10] Astrachan, Owen. Pictures as invariants. *ACM SIGCSE Bulletin*. Vol. 23. No. 1. ACM, 1991.

[11] Ginat, David. "Loop invariants and mathematical games." *ACM SIGCSE Bulletin*. Vol. 27. No. 1. ACM, 1995.

[12] Shaffer, Clifford A., Cooper, Matthew L., Alon, Alexander Joel D., Akbar, Monika, Stewart, Michael, Ponce, Sean, and Edwards, Stephen H., "Algorithm Visualization: The State of the Field." *ACM Trans. Computing Education*, Vol. 10, No. 3, Article 9, August 2010.

[13] Price, Blaine A., Baecker, Ronald M., Small, Ian S., "A Principled Taxonomy of Software Visualization." *Journal of Visual Languages and Computing*, Vol. 4, No. 3, Sept. 1993.

[14] Tillmann, Nikolai, Jonathan De Halleux, Tao Xie, Sumit Gulwani, and Judith Bishop, "Teaching and Learning Programming and Software Engineering via Interactive Gaming." *Proc. 35th International Conference on Software Engineering (ICSE 2013), Software Engineering Education (SEE)*, May 2013.

[15] Rosenthal, Anne M. "Xylem: The Code of Plants." SFNature On Assignment. 2 Jan. 2014. Web. 30 Jan 2015. <http://www.sfnatureblog.com/2014/01/xylem-code-of-plants.html>.

[16] Logas, Heather, Whitehead, Jim, Mateas, Michael, Vallejos, Richard, Scott, Lauren, Murray, John, et al. "Software verification games: designing Xylem, the code of plants." *Foundations of Digital Games*, 2014.

[17] Wang, Vincent. "'Xylem: The Code of Plants'—a (mostly negative) review." The Revolver's Notepad. 8 Mar 2014. Web. 27 Feb 2015. <http://www.pi-identity.com/blog/2014/03/08/xylem-the-code-of-plants-a-mostly-negative-review/>.

[18] Dietl, Werner, Dietzel, Stephanie Dietzel, Michael D. Ernst, Nathaniel Mote, Brian Walker, Seth Cooper, Timothy Pavlik, and Zoran Popović, "Verification games: Making verification fun." *Proc. of the 14th Workshop on Formal Techniques for Java-like Programs*. (Beijing, China), June 12, 2012.