PROCEEDINGS OF THE TWELFTH ANNUAL

CONFERENCE ON APPLIED MATHEMATICS

University of Central Oklahoma
Edmond, Oklahoma
February 9-10, 1996

DEVELOPING AN ARTIFICIAL LIFE SIMULATION PACKAGE

Dionne Bailey [1]

Ann Cheek [2]

Marcin Paprzycki [1]

[1] Department of Mathematics and Computer Science

[2] Department of Biology

University of Texas of the Permian Basin

Odessa, TX 79762

ABSTRACT: Recent publications suggest existence of a gap between computer science curricula and industry needs. We show how a software engineering project for developing an Artificial Life Simulation Package can help bridge this gap. The results of experiments investigating the influence of the artificial life parameters: initial population, food, movement, survival, and birth are presented and briefly discussed.

1. Introduction

It was recently suggested (Lawlis & Adams, 1995) that current CS curricula do not address the real needs of the industry:

> " ...industry needs software practitioners who understand the dynamics of developing and reengineering large software systems, adeptly employing techniques such as analysis and reuse." (p. 5)

They also criticize the current CS curricula for putting too much attention on creating small pieces of software from scratch. It is even suggested that in the modern CS curriculum there is no place for creating software from scratch. Similar criticisms have also been made by Jones (1995) and Khajenoori (1994).

The aim of this paper is threefold. First, to introduce a semester long software development project that is currently being used in the CS II course at the University of Texas of the Permian Basin (UTPB). Second to describe the software engineering issues encountered when three artificial life packages were combined into one. And, third, to discuss the experimental data collected using the unified package.

2. Artificial Life

Mathematical models were the primary method used for predicting a population's dynamics and state at any given time in the future. Differential equations were formulated based on simple models incorporating basic characteristics of the population and a minimal number of assumptions. Some accuracy was sacrificed by the assumptions made in these simplistic models. Artificial life simulations incorporate computational models which are not concerned with simplicity, rather a complex, detailed analysis derived from characteristics of individual members of the community. It

is only due to the computer's ability to perform fast calculations that such complex simulations became feasible (Cliff et. al, 1994, Ditlea, 1994).

Artificial life simulations involve large software systems which must be maintained and constantly enhanced to achieve greater accuracy (and to prevent the software from becoming outdated). In addition software requirements involving the user interface are an important part of the Alife software quality which needs to be augmented. In improving the simulations and user control, software reengineering consists of the analysis and reuse of the existing software packages (Jones, 1995).

3. Software Development

3.1 Project description

At UTPB the CS I course is offered not only to the prospective CS majors, but also to the students who plan to major in other sciences: biology, chemistry, geology and mathematics use it to satisfy their programming requirement. The CS II course caters primarily to the prospective CS majors. It is a relatively small course, typically consisting of one section of approximately 15-20 students. A semester long software development project was introduced two years ago in an attempt to confront the future computer scientists with a larger software artifact and address the industry needs as described by Lawlis et. al, 1995 (for the details see Paprzycki & Zalewski, 1996).

The starting idea for the project is Conway's Game of Life (Gardner, 1970). The game consists of tracing changes in the patterns formed by sets of "living" cells arranged on the grid through time. Any cell in the grid may be in either of the two states: "alive" or "dead." The state of each cell changes from one generation to the next depending on the state of its eight immediate neighbors. The four rules governing these changes are designed to mimic population change. First, a living cell dies from isolation when it has no neighbors or only one living neighbor. Dying may

also be a result of overcrowding which occurs when a living cell has more than 3 neighbors. An empty cell with exactly 3 living neighbors becomes alive. Finally, all other cells remain unchanged. This relatively simple simulation game is, during the course of the semester, extended to a larger artificial life simulation project. The project is based on group work, where the groups consist of 2-4 students – depending on the class arrangements and attrition rate. To mimic the real-life situation of software engineers the project proceeds in phases. Phase I, development of the original game of life, consists of the following steps:

1. Formulation of the software requirements,
2. Writing the design,
3. Instructor's comments on the design,
4. Implementation,
5. Testing and preparation of the User Manual,
6. Across-the-class comparisons,
7. Instructor's comments on the artifacts.

At the beginning of the process groups are formed and students study the original paper by Conway. Then they are asked to specify and discuss the basic requirements of the project (with a very strong emphasis being put on the proper development of human-computer interface). This step is followed by each group writing a design (in the form that they were introduced to in the CS I course). These designs are then commented on by the instructor.

After receiving feedback on the designs, students work on improving them and implementing the code, testing the code and writing the user manual. At this time, groups exchange their artifacts: design, user manual, and code. Depending on the size of the class, each group receives the artifacts prepared by all other groups or some subset (up to two or three) of the other

groups. Each of the groups has the following tasks: commenting on the design and user manual, verifying the code against the design, running the code and testing it, commenting on the user interface -- all tasks comprising a natural way of introducing independent verification and validation. Students are instructed to concentrate their attention on particular verification criteria, such as consistency and clarity. They are also informed that they should read the user manual from the point of view of the prospective users, so that they concentrate their attention on all things that are unclear or insufficiently explained. The improved documents are then commented on by the instructor.

Phase II, first modification of the system, is almost a repetition of the previous one, with new requirements added to the problem specification. This time students work not on the new code, but introduce modifications to their own, earlier developed, programming artifacts. This phase is then repeated as many times as there is time during the semester.

3.2 The particular project

During the Spring 1995 semester three groups of 3 students have been working on the project. Following the project description above they were able to create three packages with the following Alife parameters: initial population, aging, movement, gender, and food consumption parameters.

In populating the initial grid, the 3 groups chose different approaches. Group 1 allowed the user to specify the number of males and females to be randomly placed on the grid. Group 2 did not give the user any control over the initial population, instead 25% of the initial grid was randomly populated with males and another 25% of the grid consisted of females. Group 3 allowed the user to give the percentages of the grid to be populated with males and females independently. Groups 1 and 3 set the age of the initial population at 0 generations. Group 2 set the age randomly between 1 and 10 generations. Groups 1 and 3 set the initial food concentration at 10 units per

cell, while Group 3 randomly filled each cell with food at concentrations ranging from 1 to 10 units.

Rules for aging and food consumption varied slightly for all three groups. Groups 1 and 3 incremented the age of each organism by 1 unit each generation, while group 2 incremented the age of each organism by 1 unit every 10 iterations. Group 1 was the only group to incorporate an user changeable age limit specifying a maximum lifespan for the organisms. The rate of food consumption was the same for all three groups: each organism consumed 1 unit of food each generation. The regeneration of food in an unoccupied cell was handled by a different set of rules in each group. Group 2 allowed an unoccupied cell to generate 1 unit of food after it had remained empty for 3 generations, while Group 3 required an unoccupied cell to remain empty for 5 generations before regeneration. Group 1 allowed an unoccupied cell to regenerate 1 unit of food every generation. The occurrence of food regeneration was restricted by requiring the unoccupied cell's current food supply to range between 1 and 9 units.

Rules for movement consisted of movement probability and type of movement. Movement probability is the chance that movement will occur. Groups 1 and 3 allowed the user to change the movement probability while Group 2 set the movement probability at 100%. Group 2 provided the user with the choice between random movement and random movement with age and food restrictions. The random movement option would randomly choose a neighboring cell for the organism to enter. The age and food restriction option specified that the age of the moving organism could be at most 3 and the food supply of the organism's current cell must be less than 5 units. Group 1 chose to incorporate random movement similar to Group 2. Group 3 tried to give the organisms "intelligence" by providing them with the ability to search for an empty neighboring cell with a greater food supply. Groups 1 and 2 avoided conflict resolution since organisms did not move simultaneously, but one at a time. Group 3 allowed simultaneous movement of all organisms, so this group created a method for resolving movement conflicts. When two or more

organisms tried to move into the same cell, a strength test was applied to determine the victor. Strength consisted of an organism's age base, maturity bonus, and luck. Age Base was awarded on the age of an organism: Young (10), Adolescent (25), Primer (35), Elder (20), and Ancient (15). Maturity Bonus reflected the amount of increased wisdom acquired with age and the condition of one's reflexes which are required for battle. For the Young, Adolescent, and Primers, a maturity bonus was randomly selected from 1 through 10 which increased their strength. The maturity bonus for the Elder and the Ancient decreased their strength by an amount ranging between 1 and 5. Luck was not age dependent, thus each age group could achieve an additional amount of strength due to luck which was randomly chosen from the range 1 through 20. The organism with the greatest strength moved to the new cell while the other opponents remained in their previous cells. This strength test was applied until a triumphant winner was determined.

Restrictions placed on the rules for survival were either age-based or number-of-neighbors-based. All three groups required that the food supply for each organism could not be zero. Group 1 created an age-based rule for survival which required all organisms to be less than their maximum age. Group 2 created a number-of-neighbors-based rule for survival that restricted the number of neighbors per organism to 1, 2, or 3 neighbors. Group 3 restricted the number of neighbors of surviving organisms to 2 or 3.

All the groups incorporated similar rules for birth. Each established a 100% birth probability where 50% of the births would be male and 50% would be female. Births had to occur in an unoccupied cell with a nonzero food supply. The groups differed in the neighbor requirement for birth. Group 1 had a requirement of 1 or more neighbors, Group 2 had a 3 neighbor requirement without any stipulations on the gender of the neighbors, and Group 3 required 1 male and 1 female only.

3.3 Software Integration

During the Fall 1996 Semester a software integration project was undertaken. The merging process was accomplished using the bottom up design which begins with the individual subroutines and then builds the program up. Additional subroutines had to be added in order to give the user more control of the rules and the initial set up of the game.

For the Initial Population, the user was allowed to determine the density of the initial food supply and set the initial age of the population. Food regeneration rules were changeable such as the number of steps needed for an unoccupied cell to produce 1 unit of food. If a cell's food supply became exhausted, the user could establish the length of time required for the food in the cell to regenerate. In addition, the user was allowed to establish a maximum level of food to prevent cells from producing an infinite amount of food.

Two additional types of movement rules were added to the user's selection: Companion Based Movement and Age Based Movement. Companion Based Movement admitted a requirement on the number of neighbors surrounding a potential cell. Movement which depended on the age of the immediate neighbors of an empty cell was accessible through the Age Based Movement option. This option allowed the user to establish age groups which simulated the congregation of peers. The probability of movement was also changeable by selecting Age Based Probability or Random Probability. Age Based Probability made it possible for the user to set different probabilities for various age groups. A randomly chosen probability of movement for each iteration was the other option.

The user was allowed to manipulate the survival and birth rules by setting restrictions on age, neighbors, and/or food. For example, a requirement concerning the number of neighbors in conjunction with the age of these neighbors could be set up as a survival rule. Child Care was another option which required the parents and child to remain immobile until the child reached the age of adulthood. The age of adulthood is a user determined parameter.

The next step in software integration was establishing the effects of each subroutine on the others as well as connecting the subroutines by way of parameter lists. The final stage for the construction of the software was including all the user rules into the subroutine which executes each life generation. The life subroutine controlled the life cycles by regulating movement, survival, birth, food, and age with the rules established during the set up by the user.

A large number of software engineering type difficulties have been encountered. Most of them were related to the manipulation of a large software artifact. The final code had increased from 4,823 lines to 9,797 lines, or double in length. Merging multiple programs is complex due to various programming styles, variable names, and differing data structures. All three of the programs were different even though they progressed in a similar manner. A detailed design had to be constructed and followed so that the majority of the existing subroutines remained functional within the large software package. Varied programming styles as well as defined types slowed the merging process down. Many subroutines had to be altered so that the subroutines could be reused, but others were not salvageable and had to be rewritten more efficiently. The main emphasis of this artificial life package was expanding the user interface to create a program which could be tailored to each user. Due to the dependencies of the subroutines upon each other and the generalized user control, the number of variables and the parameter lists increased dramatically. With the broad amount of variables came the increased compilation time due to undeclared variables. One reason is misspelling since variable names were between 5 and 15 characters in length to allow for descriptive names to aid in using the broad range of variables in the large code. Debugging also became more tedious due to the number of variables and length of code. Initializing and resetting variables was a debugging error which arose most often and detection of this error was time consuming. Array indices which were out of range and infinite loops were other run-time errors that needed to be debugged. The most constraining error involved the memory

allocation of the network. Memory had to be increased in order to run the program for multiple generations.

4. Results

The final phase is experimentation with the program parameters to determine which life characteristics are most and least influential upon the survival of the population. The first experiments involve changes in initial population parameters while keeping all other parameters constant. The same process is followed for each of the other parameters: age, food, movement, survival, birth, and death.

Results from experimentation with parameters during multiple runs can be generalized by the effect that they had on the dynamics of the population. For example, manipulation of neighbor parameters for birth and survival could affect the population both by creating overpopulation or a population consisting of isolated organisms. Overpopulation and isolation had the most dramatic effect on the life expectancy of the population by destroying the organisms in 10 to 50 generations.

Food shortage could be a result of overpopulation or food parameters. With overpopulation the number of unoccupied cells decreased which kept food regeneration to a minimum, thus the food density could not sustain the population. Food parameters which lead to a food shortage are a time delay greater than 10 iterations for the production of 1 unit of food and/or a time of unproductiveness greater than 15 generations. When food shortage and overpopulation were used in conjunction, the population would survive between 5 and 15 generations. When overpopulation, isolation, and starvation were a voided by setting the survival parameters to range from 2 to 3 neighbors and the birth parameters to 2 neighbors or 3 neighbors, the population was able to sustain itself in a cyclic pattern or a stable population of about 30% occupancy. A sparse and a dense population were able to stabilize themselves under these conditions within 15 generations. Parameters such as grid size, movement, over abundance of food, and age restrictions

did not affect the length of the population survival in a noticeable way. Companion Movement did slow the dying process by enabling the organisms to remain in suitable locations, but death was inevitable since a patchy food supply developed.

5. Conclusion

Software engineering is a field which is evolving according to the needs of the users, therefore the software engineers must be able to produce applications which meet the needs of the users within a short period of time. Only if we provide students with the appropriate training will industry needs be satisfied. The presented project here is one of the possibilities for addressing such needs.

Our experiments show that even a set of rules create *ad hoc* leads to the development of an Artificial Life simulation package that relatively well matches the real population dynamics relatively well. We have found that overpopulation, isolation, and food shortage are most damaging to any population. Future work will involve an artificial life simulation based upon real population cycles in snowshoe hares. Modifications will be made to the current software package to mimic the 10 year cycles in hare populations (Krebs et al., 1995). Predators will be introduced in this simulation in order to understand the effects that they have on the hare population cycles.

Acknowledgment

This paper was prepared under the guidelines of the ARPA grant (via USAF Phillips Laboratory) F29601-94K-0046.

References

Ditlea, S. (1994). Imitation of life, *Upside*, November 1994, 6(11), 48-60.

Gardner, M. (1970). The Fantastic Combination of John Conway's new solitaire game "life," *Scientific American*, October 1970, 120-123.

Cliff, D., Husbands, P., Meyer, J-A. and Wilson, S. W., (1994). From Animals to Animats 3, MIT Press, Cambridge.

Krebs, C.J., S. Boutin, R. Boonstra, A.R.E. Sinclair, J.N.M. Smith, M.R.T. Dale, K. Martin, and R. Turkington, (1995). Impact of food and predation on the snowshoe hare cycle, *Science*, 269: 1112-1115.

Jones, C. (1995). Gaps in Programming Education, *Computer*, 28(4), 70-71.

Khajenoori, S. (1994). Process-Oriented Software Education, *IEEE Software*, 99-101.

Lawlis, P.K. and Adams, K.A. (1995). Computing Curricula vs. Industry Needs: A Mismatch, *Proc. 9th Annual ASEET Symposium*, Morgantown, 5-19.

Paprzycki, M., and Zalewski, J., (1996). Software Development Project in CS II, in preparation.