Computer Systems Lab Senior Research Project Posters 2007-2008 2nd Quarter vers.

TJHSST

May 8, 2008

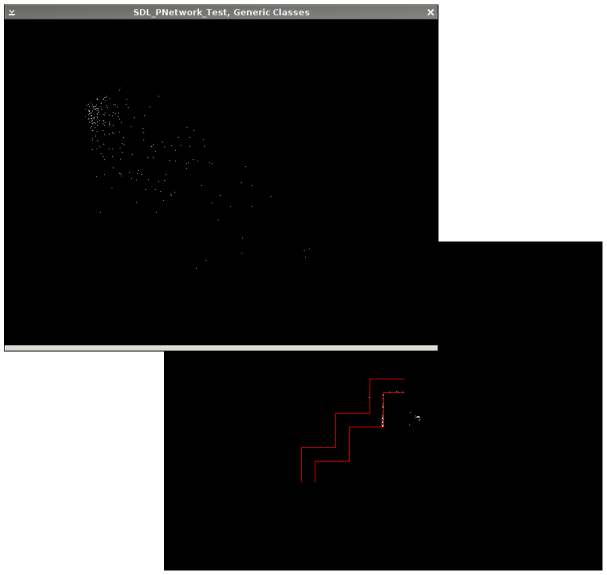
Projects
5th Period Project Posters

Artificial Intelligence:
Crowd Dynamics using Particle Swarm Optimization

Keith Ainsworth
Computer Systems Research Lab
2007-2008

Abstract

Artificial Intelligence has for long been an important aspect of computer science, but unfortunately artificial intelligence is usually computed from a single agent perspective or with multiple, but highly omniscient agents. I plan on creating an artificial intelligence engine, which works by having multiple agents, each with highly limited perspective. In order to solve tasks, they need to communicate their portions with each other through a network. Using that scheme, it will much more accurately simulate crowd dynamics, using particle swarm optimization to optimize the calculations.



Methods

To program this engine, along with the accompanying game (for graphical output reasons) I'll need C++ (and therefore the g++ compiler) along with the SDL (software digital layer) libraries, for keyboard input and graphical output, and I'll be using OOP programming (therefore the gcc compiler won't be sufficient) and PSO for optimization.

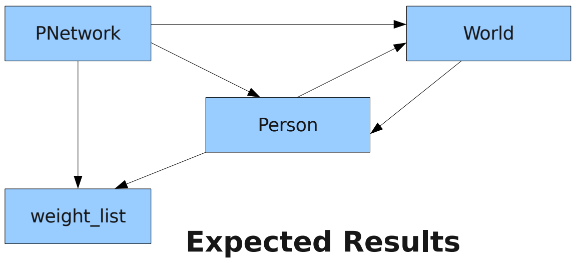
I've programmed this project so far with several debugging features and text based outputs for constant error checking. While for the final project these would be commented out for the final compilation, I plan to continue programming with those features to allow for ease of code writing and testing. Right now I'm using a series of testing shells to assess the resilience of my AI system. I have shells which print out pixels for each agent in the simulation, using SDL, which work for the regular PNetwork and ngon world classes. These automated tests inform me whether the program is doing what it should be.

Introduction

The AI engine I'm programming is implemented through C++'s object orientation. I have programmed several classes which interact in order to make a completed end project. As my engine is an agent based networking engine, naturally the first two classes are the agent, Person, and network, PNetwork classes. The main program must include an array of Persons, which is passed to the PNetwork class on instantiation. Then the main program only needs to talk to the PNetwork, as its managing the list of people from instantiation on, and will take care of the movement of the Persons.

The Person and PNetwork also utilize another class I've written, the weightlist class. This class is a set of two array based, fixed size, looping lists; one for data, the other for the data's relative weighting. The important feature of the weightlist that other pre-written container classes don't offer is a summation function. This function effectively averages the data list, based on the relative weightings, and a decay weighting that favors the more recent entries in the list. This is crucial because the communication aspect of the PNetwork has to have a way of keeping track of each Person's communications. Therefore each Person in the PNetwork is assigned a weightlist. When the PNetwork is called to iterate, it calls the communicator functions in all the Persons, and adds the message to the weightlist of every other person, assigning the weighting based on the distance the message had to travel (the distance between the two agents). Then each weightlist is summed and the results are given to their respective Persons as their new prospective direction.

Class Diagram



Expected Results

This should create a real time multiple agent based, crowd dynamics computing artificial intelligence engine, which will be applied to a game to create realistic simulations of groups of people.

Seniors next year attempting to create a program that utilizes different artificial intelligence schemes, could use this as one of them to compare and contrast effectiveness and speed.

Elementary Education in a Technology Age

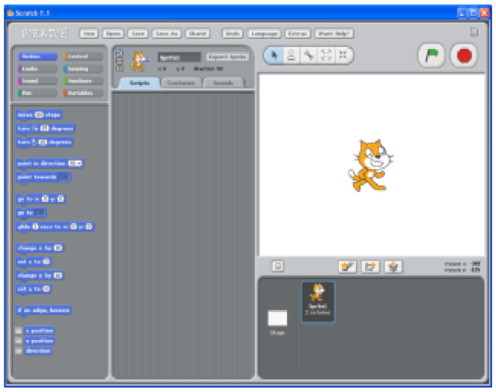
By: Gregory Gates

Computer Systems Lab Research Project 2007-2008

April 4th, 2007

Abstract

Technology becomes more advanced and more accessible with every passing day. Education should be utilizing this technology boom in teaching current students. However, this does not seem to be the case. The goal of this project is to try and implement computer programming, through Scratch, as a tool for educating students. Computer science education at a younger age becomes more and more essential as computers become more advanced and more accessible with each passing day.



Implementation

Students from first through sixth grade meet in the "Cardinal Computer Lab" at Cardinal Forest Elementary School every Thursday sometime between 11:00 AM and 2:00PM. Each class lasts for 30-45 minutes depending on the age of the kids in the class and the schedules that the teachers have laid out. The tech specialist at the elementary school and I alternate the weeks that we teach. Topics that have been covered thus far include: the coordinate axis, x-y coordinates, angles and degrees, if-then statements, basic loops, custom sprite/stage creation, and sprite interaction. Obviously, the topics include more than just computer science.

For the first three monthsMr. Allard (the tech specialist at the elementary school) and I spent a majority of the lesson time teaching the students and walking them through a simple project. The end goal is to give the students a broad category or theme (i.e. celebrations or sports) and let the students create their own projects. We have recently entered this "individual work" phase. By letting them work on their own, we hope to increase not only the students' creativity but also their independence. If the students teach themselves and work through problems on their own, they will have a better understanding of the subject matter.

Lastly, group projects will be given to promote teamwork amongst the students. These children come from many different classes and grades and they don't know each other very well, so getting along initially may be difficult. However, working in teams is an essential skill not only for computer scientists but in any job and our goal is to give these students a strong foundation for working in groups in the future.

Current Situation

Students have finally begun to work on their own individual projects, and a couple of them even claim to be finished! After taking about a two week break from any specific teaching, I introduced the Scratch-unique concept of broadcasting to the students in order to aid them in the creation of their programs. A majority of the student programs, rather than being user-interactive, try to tell a simple story or depict a simple interaction scene between a few sprites.

Background Information

The task of educating the younger generations about programming has been attempted before. The first attempt to create a kid-friendly programming language was Logo, made by Wally Feurzeig and Seymour Papert. This programming language mainly involved telling a turtle how to move around in order to make various pictures with the turtle's "pen." Since then, multiple programming environments and languages have come about to try and engage not only youth but also girls in computer science and programming such as Squeak, Alice, and Scratch (Papert, 1993).

Despite the bountiful number of tools that modern technology gives us for teaching students, little progress has been made for teaching computer science at the elementary school level. The necessary technology is present in the schools, but it is only being used to reinforce outdated teaching methods. Currently, computers are mainly being used as a medium to transfer information, much like a television. Computers have so much more potential than that. They should be used as a universal construction material, not as a TV screen. Programs like Scratch enable kids to create whatever they want to all by themselves. Children learn better by immersing themselves in whatever they're doing, rather than just listening to a teacher telling them what to do (Papert, 1993).

The goal for this project is to establish something akin to a Compute Clubhouse at Cardinal Forest Elementary School. The original Compute Clubhouse was started by the Massachusetts Institute of Technology in Cambridge in 1993 to "provide more young people with the opportunity to become digitally fluent" (Resnick, 2002). At these clubhouses, kids and older youth "become designers and creators with new digital technologies. Clubhouse members use leading-edge software to create their own artwork, animations simulations, multimedia presentation, musical compositions, websites, and robotic constructions." (Resnick, 2002)

Example Student Research Project

Student Name – Phillip Graves NEEDS ACTUAL POSTER
THIS IS A SAMPLE GENERIC POSTER
TJHSST Computer Systems Lab 2007-2008

Abstract

Provide in the Abstract section an overview of the research project. The challenges of this project are... The focus of this project is ... The goal of this project is to ... The results show that ...

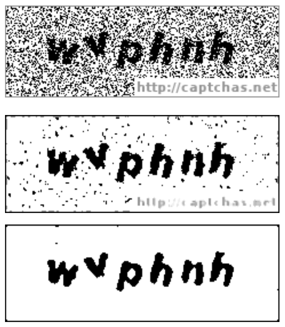
Background

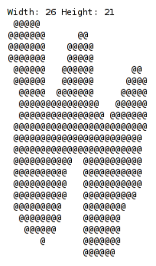
Provide in this section background information that clarifies the project's material.

You can have more than one paragraph of background information...

Paragraph 2 of background information...

What previous research has been done in this area...





```
Width: 26 Height: 21
 @@@@@
@@@@@@@      @@
@@@@@@@    @@@@@
@@@@@@@    @@@@@
 @@@@@@   @@@@@@       @@
 @@@@@@   @@@@@@      @@@@
  @@@@@  @@@@@@@     @@@@@
  @@@@@@@@@@@@@@@   @@@@@@
  @@@@@@@@@@@@@@@@ @@@@@@@
 @@@@@@@@@@@@@@@@@@@@@@@@@
 @@@@@@@@@@@@@@@@@@@@@@@@
 @@@@@@@@@@@@@@@@@@@@@@@@
 @@@@@@@@@@@  @@@@@@@@@@@
 @@@@@@@@@@   @@@@@@@@@@@
 @@@@@@@@@@   @@@@@@@@@@@
 @@@@@@@@@@   @@@@@@@@@@
 @@@@@@@@@    @@@@@@@@
  @@@@@@@@    @@@@@@@
   @@@@@@     @@@@@@@
      @       @@@@@@@
              @@@@@@
```

Procedure and Methods

The general procedure for this endeavor consists of several steps. The first step is ...

The second step is to ...

The next step is ...

The last step is ... Afterwards, it will be run to ...

The project is written in ...

Testing

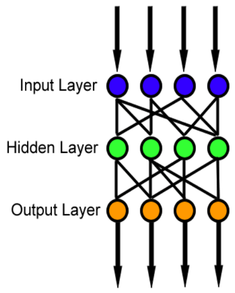
How are you testing various parts of your project in order to verify your results...

... is tested by ...

The next step is to use the outputs of ... as the inputs to ..., and then for testing ...

Results and Conclusion

What have your tests and analysis demonstrated...



Development of an OCR System

Nathan Harmata

TJHSST Computer Systems Lab 2007 - 2008

Abstract

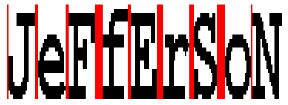
OCR (Optical Character Recognition) is a very practical field of Computer Science. Since the late 1980's, researchers have been developing system to identify text from non electronic sources, such as pictures or newspapers. The use of OCR systems has spanned from making books in Braille to sorting mail by zip code.

Background

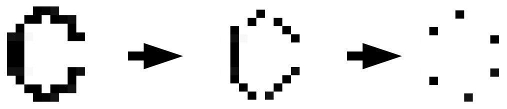
Although there are a few options currently available to the public, like Microsoft Document Imaging, most of them are either unused or not accurate enough. The goal of this project is to create an OCR system that is simple to use and can handle most formatting and fonts.

Procedures

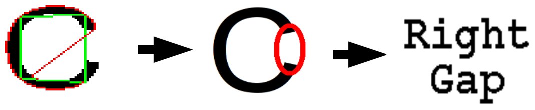
The input for the current prototype is a PNG picture file that contains text in the Courier font. Using the Java BufferedImage class, locations and colors of the pixels in the image can be determined. The program uses these to find the positions of horizontal straight lines of whitespace in the image. It pairs together lines of whitespace and ignores those so that only lines of text remain. Each line is parsed into words using a similar method involving vertical lines. After making spacing analysis, each word is parsed into letters, as seen below.



Then, each letter is converted into a form called a "CharacterModel." Each model is a collection of "attributes," currently consisting of a "SectorVector" and a "GapVector." A SectorVector is formed by parsing the image into portions that pass the vertical line test. After getting rid of unnecessary information, each portion is then transformed into a sum of line segments of different slopes.



A GapVector is simply a collection of the locations of visual "gaps" in the image. Gaps can exist on the four sides (top, right, bottom, and/or left) of the image.



Results

The result is that a letter is simplified into a few pieces of generic information. This procedure is applied to each letter of several different fonts, and information from the results is stored and averaged. Using this, a cache is created to which results from OCR analysis can be dynamically compared.

```
c SectorVector -2 3 GapVector R
```

Using Genetic Algorithms to Optimize the Traveling Salesman Problem

By: Ryan Honig

Abstract

My goal is to create a program that can solve the Traveling Salesman Problem, finding near-optimal solutions for any set of points. I will use genetic algorithms to try to find the optimal paths between the points. I would also like to expand my algorithm so that it can solve both symmetric and asymmetric problems. In the end, after I create a working algorithm that will find near optimal paths, I hope to create a graphic interface that will display the chosen points and the paths through those points as the algorithm runs.

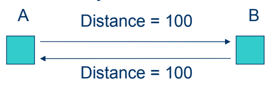
What is the Traveling Salesman Problem

Traveling Salesman Problem (TSP) - a set of points is given. Try to find the shortest path that travels between each point once and returns to the starting point

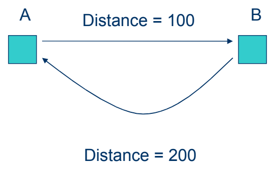
Symmetric TSP - distance between towns A and B is the same as distance between towns B and A.

Asymmetric TSP - distance between towns A and B is different from distance between towns B and A.

Symmetric



Asymmetric



Development

- •I have a genetic algorithm that creates a pool, and then uses genetic crossovers within the pool to find the best solution
- •I also have a mutation function that has a one in fifty chance of adding a variation into the pool by reversing a segment of a path, this helps to keep the pool from getting filled by copies of the same path
- •I also created a heuristic that creates a better pool than the randomized pool, although it runs much slower
- •During third quarter, I began work on converting my random pool program so that it can find near optimal solutions to asymmetric traveling salesman problems

Results

Testing the random-pool program against the Heuristically generated pool program

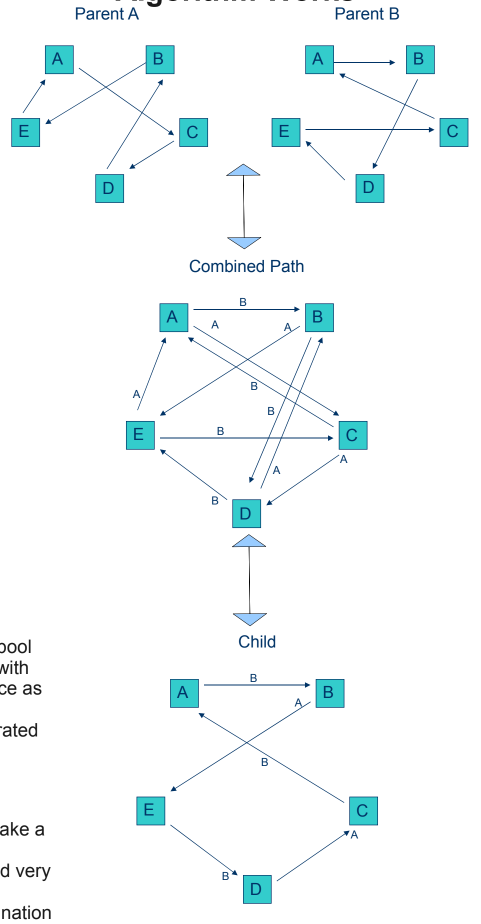
	Random Pool Program		Heuristic Program	
Data Set / Best solution	Average (of 5 runs)	Average Run time	Average (of 5 runs)	Average Run Time
A280: 2579	2780.54	1.75 sec	2729.37	3.03 sec
ATT48: 10628	12017.46	2.31 sec	12104.32	4.71 sec
BAYG29: 1610	1750.92	1.33 sec	1683.84	2.42 sec
BAYS29: 2020	2385.34	1.86 sec	2327.77	2.81 sec
CH130: 6110	6493.65	2.76 sec	6387.37	4.54 sec

•As you can see from my data, while the heuristically-generated pool program found slightly better solutions on most of the data sets, with the exception of data set ATT48, on every case it took almost twice as long to run than the randomly generated pool program did. I am currently not sure whether I will stick to using the randomly generated pool program or the heuristically generated pool program.

Background

- •Purely genetic approaches can find near optimal solutions, but take a long time
- •Purely heuristic approaches can run very efficiently, but don't find very optimal solutions
- •Many of the current best known solution algorithms use a combination of heuristics and genetic algorithms

How My Genetic Algorithm Works



3D Collision detection for N Solids in Open GL

Richard Hooepr

Abstract

Collision detection is a very useful concept, it is used in various applications from surgery to manufacturing to video game design. My project aims to create an efficient algorithm for detecting collisions so that it can be used in a gaming environment. The objects in collision will be simple solids, and multiple will be put in a space to monitor their interactions. The first step is simple 2D collisions followed by more complex 3D collisions.

Introduction

In this project, I plan to create an efficient algorithm for 3D collision detection. This project has value, because there are many different applications for collision detection, and in game development, as with all other fields, efficiency is of extreme importance. Collision detection is the concept of first detecting possible collisions, then contact, and then determining how to react to the collision. I intend to create an efficient algorithm that would detect collisions, so that the interactions of multiple solids could be modeled at once. The first step is to create a simple 2D algorithm that would model collisions as a prototype, followed by a simple 3D algorithm. This would then be optimized or redesigned, and then the number of solids in the given space would be increased, and the time taken and accuracy would be tested. The goal is to have the number of solids in space to be in the thousands, but the first benchmark would be in the hundreds.

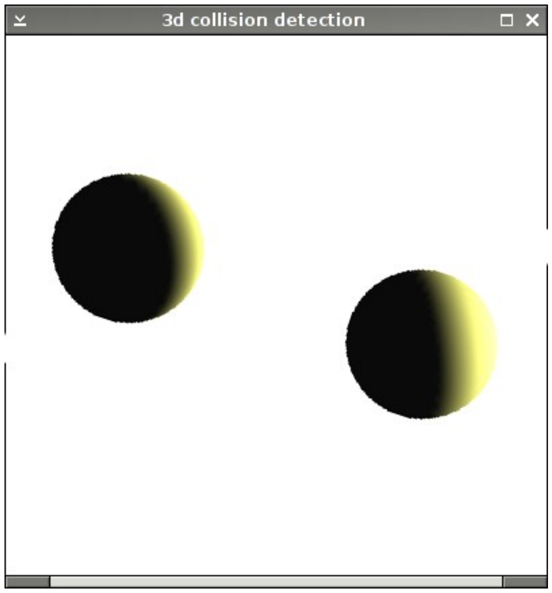
Development

This project is an effort to create a fast and efficient collision detection algorithm. Success would be considered a working algorithm that can successfully detect collisions for one hundred solids (although one thousand would be preferable). Anything less would be considered a failure.

The language used would be C using OpenGL, because C is a powerful language, and OpenGL is an easily accessible graphics library.

The workplan for the project is as follows: write a 2D algorithm, then write a 3D algorithm, then optimize the 3D algorithm or rewrite it to meet any time constraints.

So far the 2D and 3D algorithms have been completed, and the next stage is to optimize the 3D algorithm. Unfortunately the current algorithm is not very robust and only works with certain solids. This problem will have to be remedied before the project can continue.



Prisoner's Dilemma with Optional Cooperation and N Participants

Matt Lee
TJHSST Computer Systems

Abstract

This project is designed to simulate the classical Prisoner's Dilemma with a large number of participants and set options to cooperate with others or not. The purpose of this project is to allow the Prisoner's Dilemma to have variable parameters so that a variety of situations and settings could be tested. The result that is expected is a variety of simulations that will show how a specific situation can turn out when given options to cooperate, backstab, or 'join forces'.

Background

The Prisoner's Dilemma has been implemented a large number of times. There have even been competitions held to see who could make an algorithm that would maximize the payout for their specific participant. As stated by Robert Axelrod, the author of 'The Complexity of Cooperation' the best strategy for maximizing payoff is to use 'tit for tat'. Tit for tat is a strategy where the participant mimics the last move played by the opponent, which in the long run, enables the user to maximize his payout at the end of the 'game' of Prisoner's Dilemma. However, interestingly enough, when both participants initiate tit for tat, it doesn't become the optimal strategy. From here, a large number of variations have been made to the Prisoner's Dilemma, including implementing "N" participants instead of just two.

```
-   if(turn!=0)
            {

               while(run<size)
               {
                  if(run!=IDtag)
                  {
                     Boolean
  desu=(Boolean)list.get(run);
                     boolean desu2=desu.booleanValue();
                     if(desu2==false)
                        falsers++;
                     else if(desu2==true)
                        truers++;
                  }

                  run++;
               }
               if(falsers>=truers)
                  player.setDecision(false);
               else if(falsers<truers)
                  player.setDecision(true);
            }
            else
               player.setDecision(true);

            /*while(counter<size)
            {
               prisoner player2=(prisoner)
  list.get(counter);
               if(player2==player)
               {
                  break;
               }
               else
               {
                  boolean oppDeci=player2.getDecision();
                 player.setDecision(oppDeci);
               }
```

Progress

The options to choose strategies and make a number of opponents have been added to the program. The Prisoner class has undergone some changes including changing string identification to integer identification and the addition of a new method. The program itself has several variations of Tit for Tat added into it as well at this point.

Results

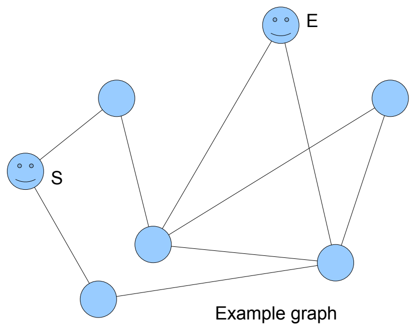
The results I expect are a variety of scenarios to be implemented by the Prisoner's Dilemma that will include "N" participants of the user's choice and option to enable morality.

Pathfinding Algorithms for Mutating Graphs

Haitao Mao
Computer Systems Lab 2007-2008

Abstract

Consider a map of an unknown place represented as a graph, where vertices represent landmarks and edges represent connections between landmarks. You have current information on the time it will take to travel between landmarks, as well as an archive about how the travel times changed through the past. You have a preset destination that you want to reach as fast as possible. Pathfinding algorithms for static graphs involve computing the whole path from start to destination, but if the weights are rapidly changing due to some extreme condition of the place, then calculating the whole path in the beginning will not be feasible. The purpose of this project is to design and compare different pathfinding algorithms for a graph whose edge weights mutate randomly to a significant extent. Algorithms may involve probabilistic analysis, dynamic programming, heuristics, genetic programming, and variations of standard shortest-path algorithms such as Dijkstra's algorithm.



Example graph

Algorithms

Define randomized distance as the distance to destination node taking graph structure into account. For example, a vertex with two unit length paths leading to the destination will be closer in this sense than a vertex with only one. We use steady-state convergence by creating a system of equations that the randomized distances should satisfy, and then approximating the solutions repeatedly until the results converge. We use dynamic programming to approximate distance to heuristically closer points first, then base calculations for farther vertices on these approximations. We use the previous states of the graph: we can use this data to develop a hashmap to approximate future mutations. We use genetic programming to find optimal values for algorithm-specific variables. We focus on sparse graphs, graphs where the number of edges is significantly less than the square of the number of vertices. The edge weights are limited to positive doubles so mutation will be somewhat controlled; edge weights that are too large will never be traversed anyway. Complexity will be limited to $O(EV)$.

Background

For this problem, the structure of the graph will be static; that is, no vertices or edges will be added or removed. Only the edge weights will be dynamic, and they must change to a significant extent for the algorithm to be effective. If the mutations are negligible, then a standard shortest path algorithm will also serve as a pathfinder. Also, the mutations should form a pattern or probability distribution. The algorithm relies upon observing previous mutation to predict future mutation, so the two must be interdependent.

In this project, several simplifications to the general problem will be made for easier simulation. In any simulation, mutation must be discretely quantified. Here, mutation will be quantified in time steps, and every edge will take one time step unit to traverse. Hence, edge weights will not represent time but instead some generic cost. In a travel analogy where edges represent roads and vertices represent cities, road condition changes due to weather would be a time-based mutation, but if each road section had a toll that changed every hour, and everybody traveled at a constant speed, then it could be accurately modeled with a mutating weight graph. Edge weights must remain positive doubles. If the mutation renders the weight too large, then it will be reverted to the maximum double value, and similarly for weights too small. Also, the algorithm design will be tailored towards mutation which is essentially random, where the edge weight mutation is only dependent on the edge weight of that edge at the previous time step. Specifically, the mutation is assumed to be independent of time, graph structure, and other edge weights.

Results and Conclusions

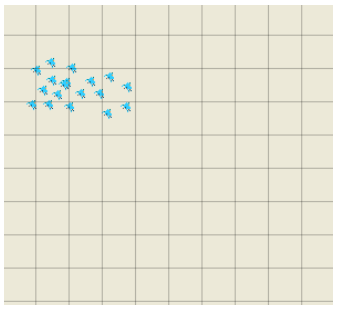
As of now, the algorithm runs a lot better than an algorithm that doesn't take mutations into account, such as Dijkstra's algorithm, would. For every case tried so far, the proposed algorithm has reached the end with a significantly lower cost than the Dijkstra would have. This difference sometimes got as high as a factor of 5, because the Dijkstra pathfinder would often get stuck in choke points because the path it found earlier has changed and one of the edges no longer exists, sometimes forcing it to go back on itself. When Dijkstra's goes back on itself, it automatically wastes two turns' worth of time and cost and gets nowhere. However, the chance that our algorithm wastes time and cost is much, much lower due to its ability to predict mutations. Sometimes it will count on a edge becoming available in order to progress, but these assumptions are completely reasonable because either the path is far away and will have a lot of time to mutate, or has been seen as drastically changing from its history data. The algorithm can also detect when it may get stuck and avoid paths that may cause it to be stuck for long periods of time. Note that these are not really results, just the current progress. Add results next quarter.

Sugarscape: An Application of Agent Based Modeling

Andy Menke
TJHSST Computer Systems Lab 2007-2008

• **Abstract:** Computer scientists have long tried to simulate things like life or human culture with computer programs. Agent based modeling is an effective strategy for this using the idea that many complex phenomena come from the interactions of simpler pieces. Sugascape is an implementation of agent based modeling that simulates human society and culture.

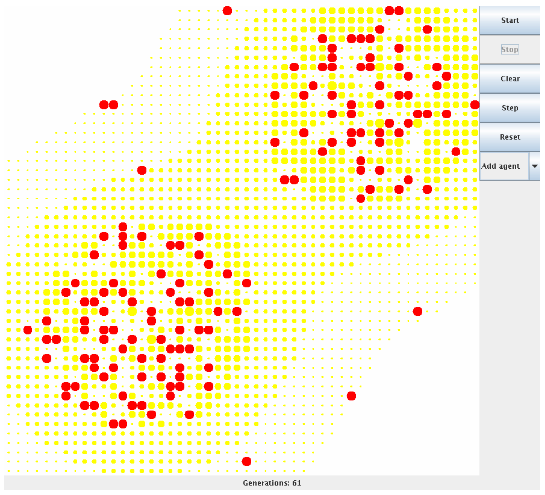
• **Background:** Agent based modeling is a technique that simulates seemingly complex relationships through the interactions of "agents" that follow simple rules. One of the first and most effective uses of this was in the creation of "boids," a program that simulates the flocking of birds. Sugarscape, in particular, recreates human society through the interactions of agents that travel around the map (scape) looking for sugar.



A Boids implementation

• **Techniques:** Object oriented programming is a technique well suited to agent based modeling. Each agent is an instance of either the Agent or Breeder class. The scape itself contains 2500 instances of the Location class, each keeping track of what goes on at a given location. The graphics are run by the ScapeG class, and the user interface has its own set of classes.

• **Conclusions:** I expect to first recreate the results given in *Growing Artificial Societies*, the book that set forth the ideas of Sugarscape. If I can do that, I will extend my project to cover areas the authors did not get to, such as warfare.



Playing God: The Engineering of Functional Designs in the Game of Life

Liban Mohamed

Introduction

Conway's Game of Life is a set of rules in a two dimensional cellular automata grid. This ruleset was specifically chosen by John Conway for the ability to create stable patterns as well as the difficulty of creating patterns which grow without bound. This difficulty was rather quickly overcome by Bill Gosper's glider gun, which opened up the ability to create binary computational devices such as logic gates. As soon as the possibility of binary computational devices in the Game of Life was discovered, it was realized that patterns could be designed in the Game of Life which could symbolically carry out computations. This project endeavors to facilitate in the design and creation of functional patterns in the Game of Life.

Abstract

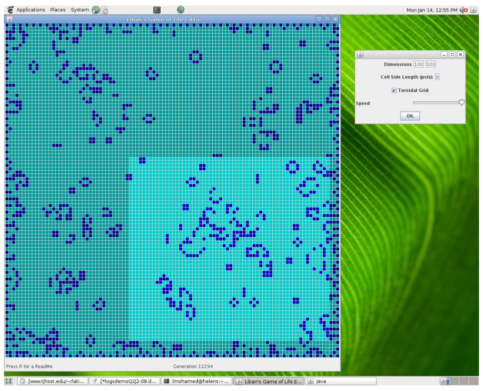
First, this project endeavours to create a flexible and powerful Game of Life interface. After that is achieved, this project goes on the create search programs for patterns in the Game of Life. Finally, this project intends to use the functionality enabled by the previous two steps to design a pattern which can be used for computation in the Game of Life. That is, the purpose of this project is to create one or more patterns in the game of life which take an input in the game of life and consistently produce an output which can be interpreted to get the right answer.

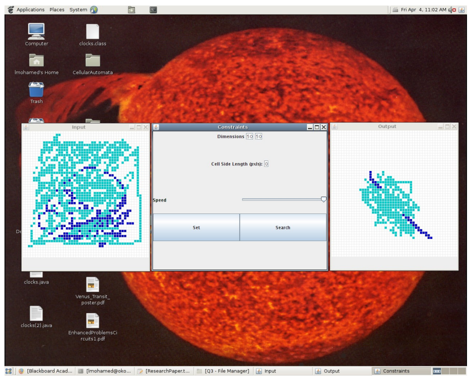
Procedures

The procedure followed in this project was to first create the programs necessary for the design and creation of functional programs in the Game of Life, and then to use these programs to actually create such a pattern.

Conclusions

Conclusions forthcoming


Dimensions 100 100
Cell Side Length (pxls): 9
[✓] Toroidal Grid


Constraints
Dimensions 50 50
Cell Side Length (pxls): 0

Procedural Generation and Terrain Rendering in a 3D Game

Justin Warfield- Period 5
TJHSST Computer Systems Lab 2007-2008

Abstract:

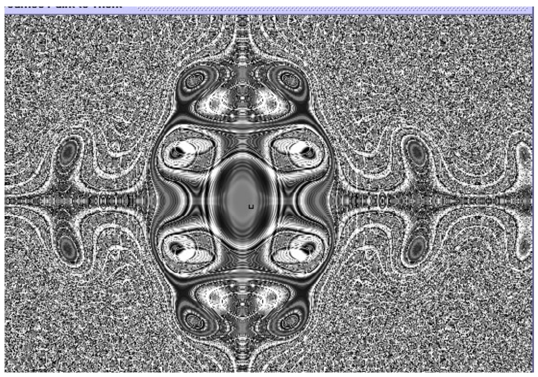
The goal of this project is to create a basic 3-dimensional video game utilizing several techniques (especially fractal geometry, multi-variable algebra, and statistical analysis) to procedurally generate terrain and game environment and render them in an efficient and effective manner.

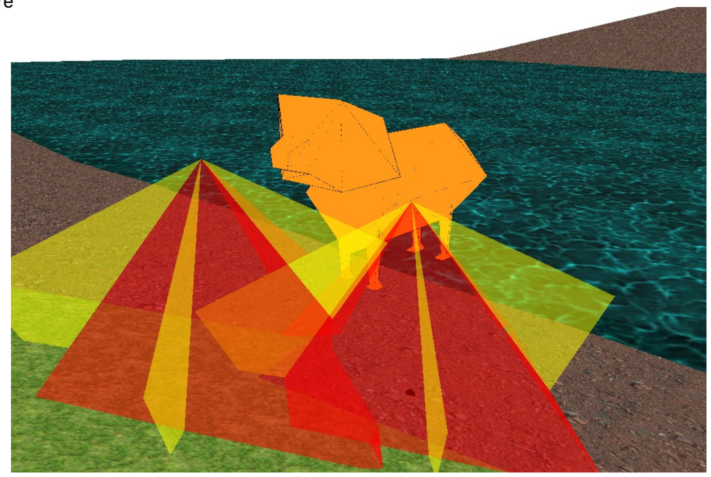
Procedures/Methods:

Using OpenGL and Python (and the OpenGL binding for Python, PyOpenGL), the first step was to create a generic 3D game. A basic framework has been created for an interactive game environment with enemies and bombs. After attempting the use of fractal geometry to generate global terrain patterns, this has been determined to be too slow (almost 10 frames per second) to generate a minimum area. Multivariable equations may be better suited to generate terrain, with pseudo-random results. The use of an equation of order 1 to determine terrain would be much faster than current techniques. Testing wouldn't be clear cut for such a program. I need to make sure what's generated is both random and realistic, which are not easily quantifiable measurements. Human testing would be most effective. Examination of different terrain equations is done using a java program that can generate an interactive height map, and the height map included in the game. Realistic environments are being added, but not yet effectively.

Background:

Many techniques are out there for creating random terrain, which seems to be the most common use of procedural generation. Fractal geometry is widely used in such algorithms. Similar techniques are commonly used to create random textures, such as cloudy skies and ground. The unreleased game, Spore, is expected to be groundbreaking in the area of procedural generation, using procedural algorithms to create 3D creature models and animations. The use of 3D equations to model terrain seems to be seldom use and research is lacking, but the speed and potential of terrain functions has drawn me to the use of multi-variable equations. Hopefully my program will be completed implementing procedural generation in a new way, paving the way for further testing and experimentation. As of now, most of my research energies have been spent learning OpenGL and its GLUT library, but I've found a few articles on gameprogrammer.com and the Intel website.





Expected Results:

Results can be presented in several ways. Comparison of various techniques would be effective. File size is also a good measurement for how much of a video game is procedurally generated (the smaller the file, the more game content is generated during game play). So far, lines of code and fps has shown that multi-variable equations are far more efficient than geometric techniques.

Use of a circular rendering area with decreasing detail with increasing area has been effective, and textures make the seams undectectable.

Conclusion:

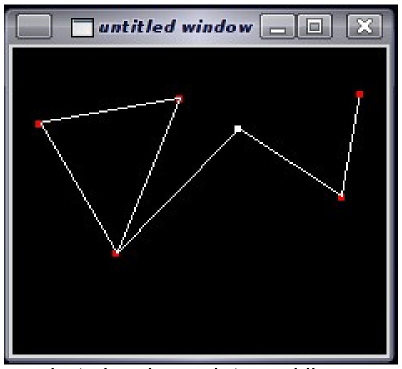
In the end, even if my ultimate goals are not realized, I will at least have contributed minor tweaks and ideas to the field of procedural generation, and hopefully even applied procedural generation to an entirely new area, paving the way to a wider range of applications. Even if my contributions are minor and don't meet my expectations, hopefully they will build upon current techniques and allow later programmers to further build upon my findings.

Interactive Geometry in 3D

Jacob Welsh
TJHSST Computer Systems Lab 2007-2008

Abstract

The goal of this project is to write a program that allows its user to create and manipulate a complex system of geometric objects in space. From a few basic object types, interesting and useful constructions can be built. This could be useful for education, mathematical or scientific research or visualization, or just for fun.

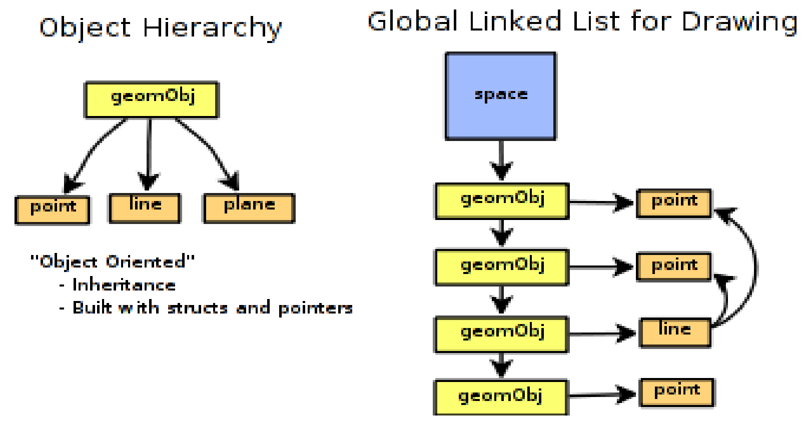


A screenshot showing points and line segments, created by clicking in various locations on the screen in Point and Line modes. The points can be selected and dragged with the mouse, and the lines are redrawn accordingly.

Background

For a while there has been software for computer assisted design (CAD), which utilizes a few basic shapes and techniques such as snapping and numeric entry to create precise, polished diagrams of a product that can then be used in its manufacturing.

A similar sort of program is used for 3D modeling, in which the user constructs polygon meshes in three dimensions: freehand; with snapping; and numerically. My program aims to be more focused on geometric objects and dynamic preservation of their relationships as some are manipulated. The leading example of this is a commercial program called The Geometer's Sketchpad. However, its interface is rather clunky, and it is limited to two dimensions. However, the fact that it is possible to build primitive 3D constructions in it illustrates the power behind the idea of geometric construction. The basic philosophy for the user interface of my program comes from the modeling program Blender and the text editor Vi.



A schematic diagram showing the internal structure of geometric objects, and the structure that stores them for display and calculation purposes.

Development of a German-English Translator

Felix Zhang
TJHSST Computer Systems Lab 2007-2008

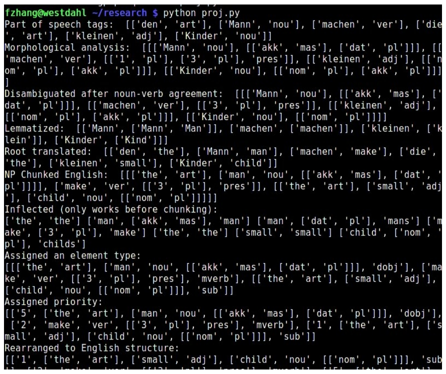
Machine language translation as it stands today relies primarily on rule-based methods, which use a direct dictionary translation and at best attempts to rearrange the words in a sentence to follow the translation language's grammar rules to allow for better parsing on the part of the user. This project seeks to implement a rule-based translation from German to English, for users who are only fluent in one of the languages. For more flexibility, the program will implement limited statistical techniques to determine part of speech and morphological information.

Background

Rule-based translation is the oldest form of language processing. A bilingual dictionary is required for word-for-word lookup, and grammar rules for both the original and target language must be hardcoded in to structure the output sentence and create a grammatical translation. Most online translators currently are based off of rule-based translation systems. Statistical machine translation is based off of a bilingual corpus, which the program uses to "learn" the language. It is much more flexible, being language-independent, but much harder to implement.

Development

The main components to a rule-based translator are a bilingual dictionary, a part of speech tagger, a morphological analyzer that can identify linguistic properties of words, a lemmatizer to break a word down to its root, a method for noun-verb agreement, an inflection tool, and a parse tree. Statistical part-of-speech tagging is implemented with a large German word corpus, with a part of speech assigned to each word. The program determines the most likely tag by checking the frequency of each tag's occurrence.

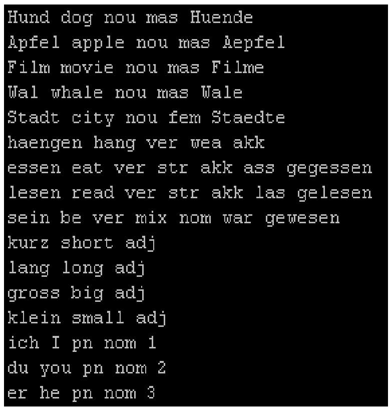


```
fzhang@westdahl ~/research $ python proj.py
Part of speech tags:  [['den', 'art'], ['Mann', 'nou'], ['machen', 'ver'], ['die
', 'art'], ['kleinen', 'adj'], ['Kinder', 'nou']]
Morphological analysis:  [[['Mann', 'nou'], [['akk', 'mas'], ['dat', 'pl']]], [[
'machen', 'ver'], [['1', 'pl'], ['3', 'pl'], 'pres']], [['kleinen', 'adj'], [['n
om', 'pl'], ['akk', 'pl']]], [['Kinder', 'nou'], [['nom', 'pl'], ['akk', 'pl']]]
]
Disambiguated after noun-verb agreement:  [[['Mann', 'nou'], [['akk', 'mas'], ['
dat', 'pl']]], [['machen', 'ver'], [['3', 'pl'], 'pres']], [['kleinen', 'adj'],
[['nom', 'pl'], ['akk', 'pl']]], [['Kinder', 'nou'], [['nom', 'pl']]]]
Lemmatized:  [['Mann', ['Mann', 'Man']], ['machen', ['machen']], ['kleinen', ['k
lein']], ['Kinder', ['Kind']]]
Root translated:  [['den', 'the'], ['Mann', 'man'], ['machen', 'make'], ['die',
'the'], ['kleinen', 'small'], ['Kinder', 'child']]
NP Chunked English:  [[['the', 'art'], ['man', 'nou', [['akk', 'mas'], ['dat', '
pl']]]], ['make', 'ver', [['3', 'pl'], 'pres']], [['the', 'art'], ['small', 'adj
'], ['child', 'nou', [['nom', 'pl']]]]]
Inflected (only works before chunking):
['the', 'the'] ['man', ['akk', 'mas'], 'man'] ['man', ['dat', 'pl'], 'mans'] ['m
ake', ['3', 'pl'], 'make'] ['the', 'the'] ['small', 'small'] ['child', ['nom', '
pl'], 'childs']
Assigned an element type:
[[['the', 'art'], ['man', 'nou', [['akk', 'mas'], ['dat', 'pl']]], 'dobj'], ['ma
ke', 'ver', [['3', 'pl'], 'pres'], 'mverb'], [['the', 'art'], ['small', 'adj'],
['child', 'nou', [['nom', 'pl']]], 'sub']]
Assigned priority:
[['5', ['the', 'art'], ['man', 'nou', [['akk', 'mas'], ['dat', 'pl']]], 'dobj'],
 ['2', 'make', 'ver', [['3', 'pl'], 'pres'], 'mverb'], ['1', ['the', 'art'], ['s
mall', 'adj'], ['child', 'nou', [['nom', 'pl']]], 'sub']]
Rearranged to English structure:
[['1', ['the', 'art'], ['small', 'adj'], ['child', 'nou', [['nom', 'pl']]], 'sub
```

Figure 3: Running version of program.

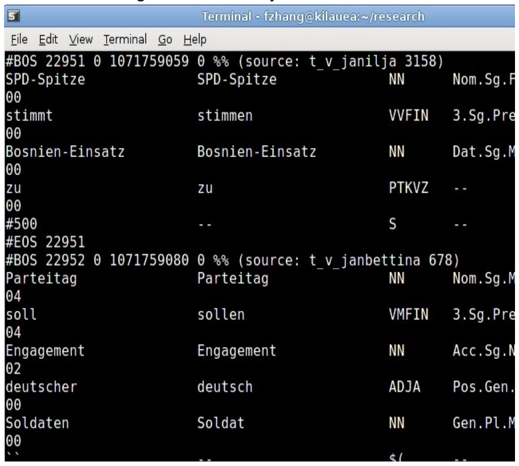
Grammar

In rule-based machine translation, parsing is the most difficult method to implement. In order to restructure simple German sentences to English ones, I assigned a priority number to each noun phrase chunk, based on where the chunk would appear in a n English sentence. The program then sorts based on priority number to restructure.



```
Hund dog nou mas Huende
Apfel apple nou mas Aepfel
Film movie nou mas Filme
Wal whale nou mas Wale
Stadt city nou fem Staedte
haengen hang ver wea akk
essen eat ver str akk ass gegessen
lesen read ver str akk las gelesen
sein be ver mix nom war gewesen
kurz short adj
lang long adj
gross big adj
klein small adj
ich I pn nom 1
du you pn nom 2
er he pn nom 3
```

Figure 1: Dictionary.



Terminal - fzhang@kilauea:~/research
File Edit View Terminal Go Help

```
#BOS 22951 0 1071759059 0 %% (source: t_v_janilja 3158)
SPD-Spitze                SPD-Spitze                NN       Nom.Sg.F
00
stimmt                    stimmen                   VVFIN    3.Sg.Pre
00
Bosnien-Einsatz           Bosnien-Einsatz           NN       Dat.Sg.M
00
zu                        zu                        PTKVZ    --
00
#500                      --                        S        --
#EOS 22951
#BOS 22952 0 1071759080 0 %% (source: t_v_janbettina 678)
Parteitag                 Parteitag                 NN       Nom.Sg.M
04
soll                      sollen                    VMFIN    3.Sg.Pre
04
Engagement                Engagement                NN       Acc.Sg.N
02
deutscher                 deutsch                   ADJA     Pos.Gen.
00
Soldaten                  Soldat                    NN       Gen.Pl.M
00
```

Figure 2: TIGER Tagged Corpus.

Results

I will run my program on a series of input German sentences, and print out the results, with a correct translation for comparison of accuracy in translation and tagging. Statistical tagging should approach 90% accuracy when each word is simply assigned its most frequently occurring tag. Rule-based methods should only function correctly with grammatically correct sentences in "normal" sentence order, with words in regular positions – Subject, verb, object.