Developing an AI Player for "Guess Who?"

Jason Pan

Period 2

Abstract

My project is to create a computerized version of the popular Milton Bradley game "Guess Who?" complete with an AI player. This involves two research areas: Natural Language Processing and Data Mining. Data mining is the analysis of data and the use of software techniques for finding patterns and regularities in sets of data. My project can be divided into two major phases: Developing the Game Interface and Developing the AI Player. My game interface consists of a matrix of buttons with pictures of the suspects and an input text field where questions can be entered. My AI's strategy algorithm will formulate questions that eliminate 50% of the suspects, which is the optimal percentage. If my program can beat an opponent at least half of the time, then I can deem it successful.

Introduction and Background

One of my favorite childhood pastimes during really long road trips was Milton Bradley's "Guess Who?", a simple two-player game. In it each player has a board with pictures of twenty different people labeled with their names. To begin, the opponents each choose a mystery person from the list of twenty. Afterwards in subsequent turns, each player asks his opponent a yes-or-no question about the Mystery Person, and he then uses the clues to narrow down the twenty suspects into the answer. A player can only guess the opponent's mystery person once, and if he succeeds he wins the game.

My project is to create a computerized version of Guess Who complete with an AI player. This involves two research areas: Game AI and Data Mining. Computer game AIs can be divided into two categories: current and classic. Current are complicated, animated games like RPG and simulation. Classics games on the other hand are the simple, strategy games that have been around before computers, like Poker, Hearts, Twenty Questions, Othello, Go, etc. Guess Who is one of the few classic game AIs that has not been done. Thus my project is original and will require new algorithms to accommodate the game's unique features. Features like interpreting typed questions and specialized file reading will prove useful for future computer games. Since my product has great entertainment worth, it will be valued by programmers and non-programmers alike.

Data mining is the analysis of data and the use of software techniques for finding patterns and regularities in sets of data. My AI's strategy algorithm will formulate questions that eliminate 50% of the suspects, which is the optimal percentage. This type of critical thinking involves data mining techniques that enable the agent to form conclusions after analyzing input data. If my automated player can defeat its opponent at least half of the time, then I deem it successful.

Data mining, also known as knowledge-discovery in databases (KDD), uses computational techniques like statistics and pattern recognition. Although it is usually used in relation to analysis of data, data mining, like artificial intelligence, is an umbrella term and is used with varied meaning in a wide range of contexts. Its official definition is the "nontrivial extraction of implicit, previously unknown, and potentially useful information from data." This encompasses a number of technical approaches, such as clustering, data summarization, learning classification rules, finding dependency networks, analyzing changes, and detecting anomalies. My project however implements an elementary form of KDD; one that does not require complicated equations or extremely sophisticated search algorithms. Developing data mining software often involves corporate funding in the millions and teams of programmers working in cooperation with project deadlines of years. Because I work solo and only have the length of a year, my project can at best produce only a crude but effective data mining function.

Data mining has many practical applications in the fields of science and mathematics and especially in the business world. Retail companies make a large use of KDD so that they can identify patterns among customer purchases and thereby reorient their marketing strategy. Developed in the 1960's, data mining really gained ground in the 90's but is still a relatively new field. New purposes are being discovered every year as more sophisticated algorithms applying those principles are programmed. I hope to make a minor contribution by applying it to turn-based computer games, which according to my research has been largely undone.

Research on game structures was limited because most games' code was close sourced. Thus I had to largely develop the two player version on my own from scratch. My game format is largely inspired by Battleship, which I programmed during my summer computer science class. I utilized several matrices to store the suspects' images and attributes, which the user can access via a grid of buttons. Also in common is that they are both turn-

based games where the player tries to locate his opponent's secret target. Privacy must be maintained and only one player can look at the screen at a time.

The guessing game closest in functionality to Guess Who would be Twenty Questions. Like GW, 20Q is a two-player, turn based guessing game that uses probability and deduction. However the AI agent functions differently. 20Q's AI involves a machine learning technique called the expert system. During a game, the AI maintains a list of questions and as it iterates that list, it eliminates possible suspects based on the user's responses. After each game, the AI learns from its mistakes and expands its knowledge database. However, my game AI will actually formulate questions instead of merely regurgitating them. I hope to develop a data mining techniquethat searches for a shared pattern among the suspects and then generates a yes-or-no question, that no matter how the user answers will eliminate half of the list.

Procedure

I decided to code my game in Java because of my substantial experience with it and the language's conveniently built-in GUI. My game is very low budget and the only resources I needed were a java compiler and internet access. This research paper was written in LaTex, and the poster and PowerPoint presentation was created using OpenOffice on Linux machines.

My project consists of four files. To store the suspects I wrote a special Person class whose fields were physical attributes. The object has a constructor, modifying methods, retrieval methods, and an equals method. GIPanel.java is the main file, and it comprises of the two-player game interface. An extended one-player version is AIPanel.java, which in addition contains the AI opponent. Both of these files are saved in JPanel form, which Driver.java then loads up and runs.

GIPanel.java, which contains the bulk of my code, is organized in an evenly-allocated fashion. The code is divided into around a dozen small methods of comparably equal size, which made it easier to debug. The one exception is the constructor, which requires no input and sets up the board. All important variables are declared globally so that the methods will not require numerous arguments.

My project can be divided into three iterations, each of which required

one school year quarter. The fourth quarter was spent on completing this research paper and preparing a poster and PowerPoint presentation. In the first quarter I concentrated mostly on research and organization and did very little coding. I studied the structures of similar computer games and existing data mining techniques to design my program. I also researched GUI methods to develop the game interface. I decided to mold my game's structure after Battleship and its functionality after Twenty Questions. Most of my work during this time was mentioned in Introduction and Background.

Phase 1: Game Structure and Logistics

In this iteration I developed the actual game infrastructure and made a functional two player version. There were some delays achieving this, as this undertaking was much more intricate than I had planned. Defining the ActionListeners and ButtonHandlers and setting the labels required only some background reading. Setting up the scoring system, matrices, and turn-based apparatus needed just thoughtful time and patience.

The most complicated task of this phase was coding the input text field. This alone can constitute its own research project and required careful planning and persistent effort. In theory the user should be able to enter a really complicated question that the game mechanism then interprets and reacts accordingly to. However realizing the immense complexity of such an algorithm, I more realistically toned down my expectations to a maximum of two traits, with some variability by the addition of "and," "or," and "not." All questions must be phrased precisely using a special jargon for the attributes. The program reads this as a stack and if there is an error notifies the player.

Another tedious task was creating the twenty suspects. I largely copied the original game's design and infringed on Milton Bradley's copyright. I scanned the characters' pictures from the actual game board and used Microsoft Paint for formatting. Using the special jargon mentioned earlier, I coded the suspects attributes into a text document, separating each trait with a comma. When running, my program reads the txt file, deconstructing each line into nine traits, which is then entered as arguments into Person's constructor to create a unique Person object. The nine traits are nose size, gender, shirt color, eye color, hair color, glasses, hat, beard, and baldness. The Person objects are stored in the *people* matrix, which is continuously referred to during game play when questions are entered or a suspect is chosen.

Phase 2: AI Player Research and Development

The most innovative and difficult part of my project was developing the AI player. Because Phase 1 ran overtime and encompassed two quarters, I was forced to quickly assemble the AI agent. However I did not have time to integrate it into my Game Interface, so users will not be able to play against it firsthand. I designed my algorithm mainly through trial and error. After reading up on the theory behind data mining, I was able to comprise my own coarse interpretation. Using a depth-first search, the AI iterates through the possible suspects and formulates a question that satisfies the set constraints. To test whether my algorithm works, I had it analyze a game board state, to which it responded with its question. I then calculated what percentage of suspects was eliminated by that question.

As a control I developed an AI agent that makes its decisions using a random number generator. Choosing a mystery person and formulating questions are all decided on a random basis. Needless to say this AI agent was ineffectual. Its inability to "think" made its strategy arbitrary and easily beatable. However this primitive prototype did accomplish its purpose, which was to serve as a basis of comparison against my "thinking" AI. The success rates of my AI and the random AI are compared in Results and Conclusions.

In-depth Explanation of Game Structure and Functionality

The goal of Guess Who is to narrow down a group of 20 suspects through yes-and-no question asking to identify the mystery person. Each suspect is of the Person class, which means s/he has a name, picture, and physical attributes. Based on the pictures, the player can enter a question about the mystery person's physical attributes. The user can interact with the game via buttons and a text field. I divided my game interface into three panels.

The top panel's purpose is to provide information. It consists of a label and a row of attribute buttons. The label displays the opponent's question from the previous turn and how many suspects s/he has left. The user clicks on the buttons to formulate a syntactically-correct question.

The middle panel consists of a 5x4 grid of buttons. Each button has a

picture of a suspect. The player clicks on the grid in the beginning of the game to choose his mystery person, and again at the end of the game to guess his opponent's. Corresponding to the grid are several 5x4 matrices. Each suspect has a unique 2D coordinate, which links the grid and the matrices. For example coordinate [1,4] on the grid is the button through which suspect Susan can be selected. Coordinate [1,4] on the Person matrix contains Susan's Person object...coordinate [1,4] on the ImageIcon matrix contains Susan's picture...etc. There are two additional integer matrices, memory1 and memory2. Memory1 keeps track of which suspects player1 has eliminated, and memory2 does likewise for player2. Value 0 is the default, 1 indicates eliminated, and 2 identifies mystery person status. The grid and the matrices are all created at the start of the game based on input from file suspects.txt, where each suspect's unique attributes are stored.

The bottom panel consists of an input text field and three buttons. The text field states the turn number and which player's turn it currently is. In the beginning of the game the user enters his name into the field. For the rest of the game the user enters his question. Afterwards the user clicks the Done button. If the question is valid, the appropriate suspects are eliminated which is illustrated by the corresponding buttons being disabled. The screen then turns yellow and informs the next player that it's his turn. This transition screen is a vital feature since it protects each player's privacy from the other. The next player then clicks the Ready button to indicate his readiness to begin his turn. The game continues like this until one player has identified his opponent's mystery person. To reset the game, a player clicks the third button, the Reset button, which activates the *setBoard* function.

Natural Language Processing

Although I did not plan so, my project involves elements of Natural Language Processing. The nature of Guess Who is that the user enters questions, which my program then has to break down and interpret. To achieve such a task I wrote a very primitive natural language processor. The user can only enter up to five words into the input text field. This includes a maximum of two traits and one conjunction connecting them. To prevent any input typos, I created a panel of buttons that the user presses to formulate his question. The syntax of the input must be precise, and I added a throw exception clause

to catch any errors. If the format of the question is not exactly correct, the program will not accept it and asks the user to rewrite his query.

Supposing the question is correctly written, my *parse* function then breaks it down and converts the String into an array of words. The *interpret* function then does the rest of the work. Using a switch statement the method is divided into three paths: whether there is an "and," "or," or no conjunction at all. The absence of a conjunction means that the user was asking about only one trait. The *process* function is then called, which iterates through the list of suspects, identifies those that don't fit, and makes the according changes to the memory matrix. The addition of an "and" or "or," only makes the task a bit more complicated. It can be inferred by the presence of a conjunction that the user is asking about two traits. The questions is the broken down into two parts, the substring before the conjunction and the substring after, and each piece is entered into the *process* function separately. When iterating through the list of suspects, *process* checks if both traits are satisfied, and based on the Boolean response modifies the memory matrix. Any error during this very complex algorithm alerts the throw-exception-catch statement, which promptly rejects the input and asks the user to re-enter it.

Artificial Intelligent Agent

I have been an avid player of Guess Who for at least four years. The beauty of the game is its infiniteness of possibilities. I have never seen the exact same game play duplicated twice. In the real world, questions can be as long and complicated as desired. However constrained to the still primitive reality of today's computers, there are limitations on the players ability to compose creative queries. Still this is only a minor hindrance since most players can compute only up to two or three traits. Usually the safest strategy is to ask questions that for yes or no will eliminate half of the suspects. Some bold individuals dare a move whose breakdown lets say is if "yes," eliminate 70% of the suspects, and "no," eliminate 30%. This is unnecessarily dangerous, since if "no" is the response, the player will suffer a major setback. The proper strategy is to make steady gains, every turn eliminating the volume of suspects by a factor of two, and wait for the opponent to make a sloppy mistake and fall behind. It is only appropriate to gamble a risky, uneven

question when the opponent has much smaller volume of suspects than you do.

The AI strategy algorithm is simple. Using a depth first search, iterate through the list of traits. If any trait is shared by 45%-55% of the current roster of suspects, that trait is formed into a question. If not, the AI moves on to plan B. The percentage each trait eliminates is recorded in an array. On a systematic basis two traits are chosen, and every possible combination is run through it. These include adding a conjunction or prefixing a "not." Each scenario is checked on whether the condition is met. For "and," the previously-calculated percentages of both traits are multiplied together. For "or," they are added up. Adding a "not" subtracts the traits percentage from 100. During all of this, any question whose range falls between 35%-65% is stored onto an array in sorted order. In the event that no question matches the original description, the next best question is popped out and used. Lastly if all else fails, a random question will be selected. In the event that opponent has 2/3 or less of the suspects of the AI, in order catch up the percentage constraint will be temporarily expanded to 25%-75% to allow a possible catch up.

When choosing a mystery person, some suspects clearly have an advantage over others. The same principle of asking questions applies to choosing a mystery person. Select a person whose has the most traits the most in common with the most people. For example, dont choose a black character because there is only one black person in the game. All an opponent needs to ask is "Is your mystery person black?" and its over. The best person to choose is a white bald male wearing a blue shirt. It takes a minimum of five turns to arrive at the answer. Using this principle, each character is given a weight depending on how common its traits are. The AI will still choose randomly to remove any element of predictability, but the agent will be inclined to choose smartly.

Results and Conclusion

Truth be told, my endeavor has been less of a research project than a development project. Instead of experimenting I have spent most of my time creating. From the beginning I already had an idea of what the final result would be and the only real experiment was whether my project would work or

not. Thus as a result I don't have much to data to show. My program is not designed to produce output for analysis but to create a workable, usable final product. Because I was not able to complete an interacting AIPanel.java, the users could not test my AI, which meant I couldn't compile data on the game-playing prowess of my AI against real world human opponents. However I obtained the next best result, which was comparing my AI's question forming ability to that of a random AI. As mentioned in Procedures: Phase 2, I measured their performances by how close their elimination rate was to 50%, which is the optimal percentage. After ten trial runs, my data mining AI formulated questions that on average eliminated 34% of possible suspects, while the control random AI averaged a mere 29%. I am quite pleased with this result, which provides evidence that my AI is making decisions on a logical basis. Though I was never able to have my AI compete against human players, I am sure it would have yielded them no easy victory.

Though my project has not turned out as expected, I still deem it a success. I started the school year with no prior experience with Linux and HTML and a shaky at best knowledge of Java GUI. Much of the first quarter was spent struggling to learn basic technical skills. Most of the benefit that has come from this endeavor is not derived from the end product but from the process of developing it. This has been my biggest and most complicated programming project ever, and I learned to integrate multiple intricate files and coordinate complex functions. The scoring system, text box, button grid, file reading, and suspect matrix all work in cohesion to form one operational unit. Every button activates several methods, which return values to each other and produces a net change in the game interface. The sheer magnitude of my projects complexity left plenty of room for bugs, not all of which I was able to fix. Most of my project's components are working and I can demonstrate them, but petty errors in two or three have prevented me from being able to exhibit a sleek finished product. However all is not in vain; the most valuable lesson I learned is how to organize and manage a large project. While this year was not as successful as I had desired, it has provided a learning experience I can apply in the future.

Since the project is not complete there is much area for future effort. An obvious addition would be integrating the AI into the game interface to create a snazzy one-player Guess Who game. Another would be applying more sophisticated data mining algorithms that can devise more thorough questions in much less time. My program lacked many basic features of data mining,

like classification, which is grouping data together according to similarities or classes, and probabilistic approach, utilizing graphical representation models to compare different knowledge representations. My program also used the most inefficient search algorithm, selection sort, whose run time is $O(N^2)$. More efficient techniques like heap sort, merge sort, or quick sort would have decreased the run time, though negligibly, since the run time is already quite small.

I would also like to expand the text box's capabilities. Natural language processing was an unexpected yet intriguing area of my project, and if I could do everything over again I would have focused my project solely on that topic. I wish I had more time to develop more sophisticated *parse* and *interpret* functions that could take in questions of any size or complexity, like a calculator does. This would require a search method that could navigate its way through a maze of parenthesis and prioritize which segment of the question to process first. I have no clue how to accomplish this, but I know its doable since a regular TI accept really long equations. This is a topic I would definitely like to study further, and will be the subject matter of any future research projects I pursue next year at the university.

I have several recommendations on how to improve the Computer Systems technology laboratory so that future generations will find it a more useful and enjoyable experience. First would be encouraging group projects. Group projects have the potential to achieve much more significant progress because more heads are better than one. Plus students will get first hand experience in collaborating to write one large program, something that TJ doesn't prepare students enough for but is how all software is written in the real world. Also, working in groups cuts down on work load and repetition, making the class more fun and effective. Another suggestion is to give less blackboard assignments. They create an unnecessary burden of busywork for the students and detract time from actual coding. A more helpful way to check on students is for the teacher to meet with each student weekly, discussing that individual's progress and making sure that s/he is on task. That way the student will be clear on what s/he is working on and with constant one-on-one motivation will be less likely to succumb to laziness. Despite these difficulties, my overall experience this year in the Comp Sys tech lab has been both pleasant and fruitful.

References

For additional information, visit my website www.tjhsst.edu/ jpan

20Q <http://www.oozinggoo.com/20q>

AI FAQ <http://www.guessmaster.com/aifaq.asp>

Applications of Data Mining <http://www.the-data-mine.com/bin/view/Misc/ApplicationsOfDataMining>

"Classic Games" AI <http://www.gameai.com/clagames.html>

Data Mining and Discovery <http://www.aaai.org/AITopics/html/mining.html>

Data Mining Techniques <http://www.statsoft.com/textbook/stdatmin.html>

Machine Learning in Games <http://satirist.org/learn-game>

Statistical Data Mining Tutorial <http://www-2.cs.cmu.edu/ awm/tutorials>

What is Data Mining? <http://www.anderson.ucla.edu/faculty/jason.frand/teacher/technologies/palace/datamining.htm>

What Is Data Mining and What Are Its Uses? <http://www.darwinmag.com/read/100103/mining.html>