System Design Document

CMSC 471
Fall 2004

Phase: Design

Team: Core-Dump

-
Spencer Shimko
-
Doug Ingalls
-
Brian Williams

Table of Contents:

1. Introduction........................ 3
  1.1 Purpose....................... 3
  1.2 Scope......................... 3
  1.3 References.................... 4
  1.4 Overview...................... 4
  1.5 Constraints................... 4

2. System Overview..................... 4

3. System Architecture................. 5
  3.1 Architectural Design Diagram.... 5
  3.2 Architectural Alternatives...... 6
  3.3 Design Rationale................ 6

4. Data Design......................... 6
  4.1 Server Required Functions....... 6
  4.2 Additional Functions............ 7

5. Interface Design.................... 9
  5.1 "Peek" Interface................ 9
  5.2 "Block" Interface............... 9
  5.3 "Shoot" Interface............... 10

6. Testing............................. 10

7. Time-line............................. 10

1. Introduction

1.1 Purpose

This document describes the approach team Core-Dump is taking to implement an intelligent game playing agent. It details the methods used to play a successful game of Battleship-471. Success, in this context, refers to the game playing agent's (hereafter referred to simply as "player") ability to maximize utility through a series of peeks, shoots, and blocks. The reader is expected to have an understanding of Lisp programming, data structures, and rudimentary understanding of A.I.

1.2 Scope

For this project two separate software components are utilized. The first is the game-board generator which has been provided and is described in the project description. This generator builds the boards, informs the user of board parameters, and acts as a game master. The second component of this project is the player implementation. The "player" will receive a general description of a game-board from the generator discussed previously. The player will enter either a one-player or two-player mode. Actions are then taken by the player that will be either: peek, shoot, or block. In the one-player mode the goal of the agent is to maximize utility. In the two-player mode the goal is slightly modified. While the player maximizes its own utility it will try to minimize the utility of it's opponent. The exact sequence of actions depends on a combination the current game-state, the type of game (one or two player), and the type of intelligence the player is utilizing at any given point. Team Core-Dump's agent will utilize Bayesian reasoning, planning, and learning to reach the goal described above. The player will learn the game-board generator by playing many games before the tournament. During game play the agent will attempt to learn about its opponent. The information learned will be stored in a Bayes net. The benefits of implementing a learning agent utilizing a Bayes net far outweigh the costs in implementation and runtime. The drawback to implementing is learning agent is that it relies on predictability. If the game-board generator was not

predictable or the opponent played in a random fashion this implementation would not fair as well.

1.3 References

Project Description:

- www.cs.umbc.edu/courses/undergraduate/471/fall04/hw/project.html

Web Material:

- www.cs.umbc.edu/courses/undergraduate/471/fall04/slides/
- www.aima.eecs.berkeley.edu/slides-pdf/

Textbooks:

- Russel, Stuart and Norvig, Peter.
  "Artificial Intelligence: A modern Approach."
- Graham, Paul.
  "ANSI Common Lisp."

1.4 Overview

The rest of this document will be dedicated to clarifying the statements made in section 1.2 above (Scope). Specifically, a more descriptive overview will be presented followed by a discussion of the architecture and data design. Finally this document will briefly discuss the human interface, testing, and time line.

1.5 Constraints

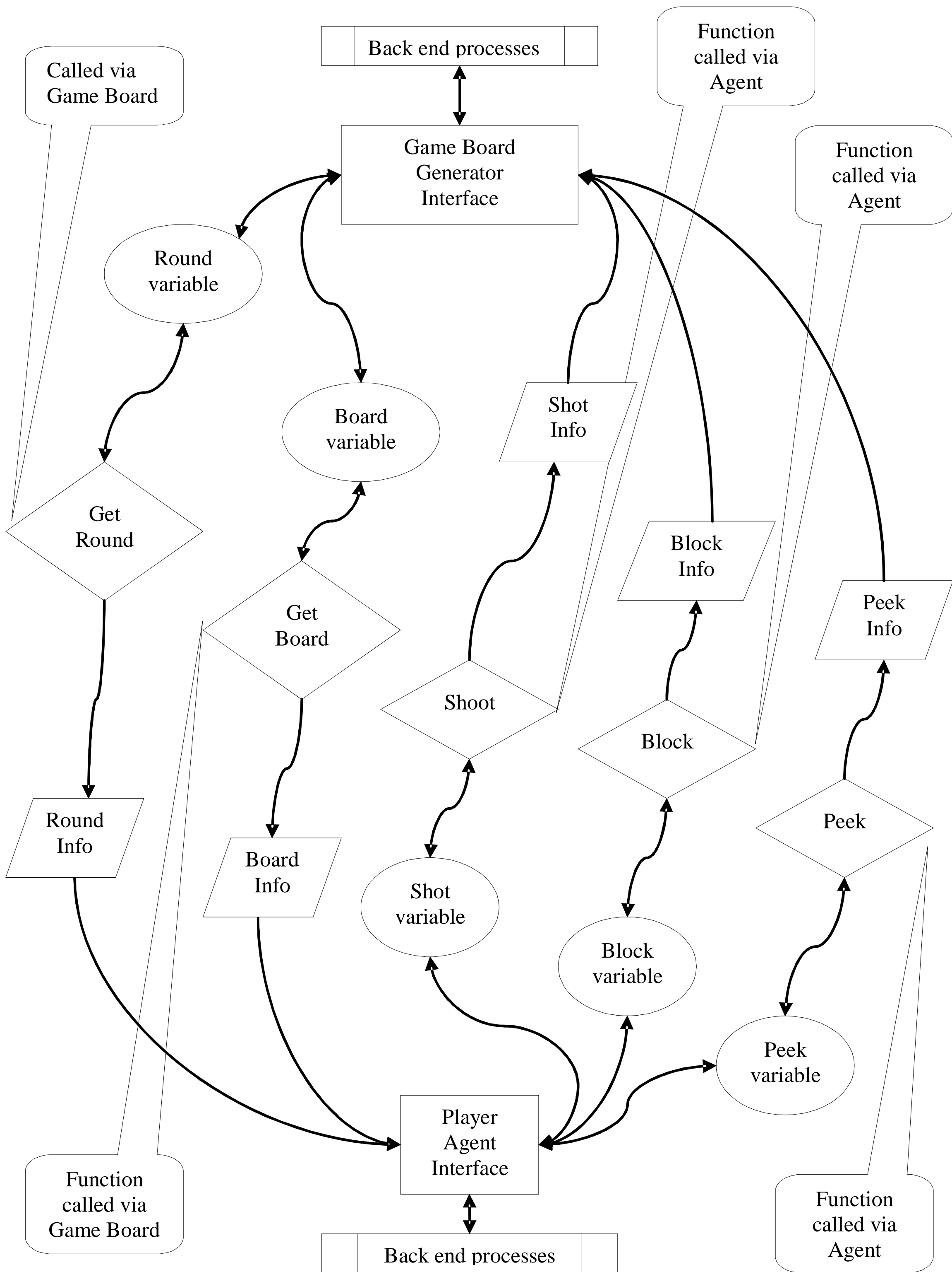
The agent is only informed about a few details such as the game-board size, ships, and the actions taken by an opponent. The agent will not be made aware of the result of actions taken by an opponent. The ultimate human goal of this implementation is to win a tournament against other players. The agent goal is to maximize utility points this is to aid in winning a tournament against other possibly intelligent agents. This constrains what we can implement minimally (in order to be successful in a tournament) and the time limit imposed could limit the complexity.

2. System Overview

The aim of this project is to develop a agent that is able to play a modified version of Battleship. The artificial intelligence class has split into groups and each group is responsible for developing an agent to play this game in an upcoming class wide tournament in December. The description is written to allow teams to control the development of their own agents. The system that will be implemented by team Core-Dump will use a combination of Bayesian reasoning, planning, and learning. This will result in a winning combination for playing Battleship 471 against the agents written by other teams.

3. System Architecture

3.1 Architectural Design Diagram


Back end processes
Called via Game Board
Function called via Agent
Game Board Generator Interface
Function called via Agent
Round variable
Board variable
Shot Info
Get Round
Block Info
Get Board
Peek Info
Shoot
Block
Round Info
Peek
Board Info
Shot variable
Block variable
Peek variable
Player Agent Interface
Function called via Game Board
Function called via Agent
Back end processes

3.2 Architectural Alternatives

Other alternatives that were considered by team Core-Dump consist of mainly knowledge based agents. The problems that we have with knowledge based agents is that they will have large amounts of data with no direct ability to assess probabilities of ships. It will be computationally more efficient to hold all probabilities in a two dimensional array and make decisions based on that. With a purely knowledge based agent and an opponent based on uninformed searches our agent is likely to perform poorly. We also discussed the use of LISP vs. C++. We thought of ways we could code this problem in C++ and how the problem would be solved with a functional language. We found that C++ would provide the same abilities as LISP, and we are much more familiar and partial to C than LISP. The advantage with LISP is it is easy to do incremental design and easy to design using bottom-up programming.

3.3 Design Rationale

We chose a Bayes net to hold all of the current information in the game to simplify all decisions on shooting and peeking. This allows us to collect all information on the game board and hold it in a simple data structure. If we only used the Bayes net, we would not be able to store the information on the histories of the previous games, so we will be storing this data outside of the Bayes net and then analyzing it and incorporating the data into the Bayes net at the beginning of every game. This minimizes the data analysis during the actual game play, making our agent run faster.

4. Data Design

4.1 Server Required Functions:

(get-board board-height board-width ships)

Description: This function is called by the server to transfer board information to our player when the game

is set up. This function will create the game-board needed for our agent. It will initialize probabilities used for Bayesian reasoning based on learned data stored externally. This will be done by calling (load) and altering probabilities based on the history of game boards generated.

(one-round opponent-action)

Description: This function is called by the server to let our program play for one step in each round. In this step, our program will, evaluate any moves made the opponent, update our Bayes net accordingly, draw conclusions based on the information represented by the Bayes net, and take actions based on those conclusions. To perform this task it will call the (update-opponent round) and (update-core-dump round results) which are described below. To actually decide which move to make this function will call (calculate-move) and take actions returned by this function.

4.2 Additional Functions:

(update-opponent round)

Description: This function will update our Bayes net used to plan our agent's moves based on the opposing player. It will take an agents move (defined in "one-round") as an argument. This function will maintain a history of opponent moves and will update probabilities based on what type of action trend the opposing agent is taking. For instance, if the agent peeks in one move and shoots during the next move our agent will adjust the probabilities of that location and it's surrounding squares.

Returns: nothing

(update-core-dump round results)

Description: This is the friendly version of the function described above. It will track the moves made by team Core-Dump's agent and will buffer the results

of those moves. Shots will not be taken immediately based on these results for two reasons. Our agent will avoid the type of predictability we are utilizing in (update-opponent) to learn from our enemies moves. Also by buffering the results of our moves we will allow our Bayesian net to contain more reliable data before shooting. Peek trends will be followed more closely since an intelligent agent will not weigh peeks too heavily since peeks may be inaccurate.

Returns: nothing

(calculate-move bayes-net)

Description: calls the (next-????) function family to decide which moves to make. This will also responsible for managing the shoot buffer. Any shots returned by (next-shot) will be buffered along with a number indicating the priority of the shot. This priority will be returned by (next-shoot) along with the square to target. Later we may decide to ignore shots under a certain priority to avoid wasting utility.

Returns: list of moves to pass to server

(next-peek bayes-net)

Description: Use the Bayes net to plan the best place to peek.

Returns: list of position (x, y) or nil

(next-shoot bayes-net)

Description: Use the Bayes net to predict the next place to shoot but additionally calculate a priority for the shot based on the Bayes net. This priority will indicate how “strong” the shot is. Since the agent will buffer shots we must provide a mechanism for firing shots that are likely to result in a higher return first. Using this method we don't run the risk of the

opponent firing on that location before we can remove the shot from our buffer.

Returns: list of position/priority (x, y, pri) or nil,

(next-block bayes-net)

Description: Decide the next place to block. This will always return a location since it has no utility. This will be based on what we have learned of our opponent.

Returns: list of position (x, y)

(save game-state)

Description: Saves a game-board filled in with results of a games worth of turns. This should indicate ships locations based on our end Bayes net from each game.

Returns: nothing

(load game-state)

Description: Load our previous history of game stats

Returns: A list of previous game-boards

5. Interface Design

The only functional interfaces that the Player agent can access are the three functions provided by the game board generator. These three functions are "Peek," "Block," and "Shoot." During the course of a game, the Player agent will make an educated decision on what the next turn will entail. Imbedded in the "play game" functionality of the Player agent, an interaction with the game board generator will occur when these three functions are called. The function prototypes of these functions are as follows.

5.1 "Peek" Interface

(peek x-co y-co cost)

Description: The functions take's three arguments. A "x-co" represents the x coordinate of the game board plain. A "y-co" represents the y coordinate of the game board plain. And the desired "cost" a number between 1 and 5; which represents the desired value of peek.

Returns: The function returns a data type. The data type is a "ship-name", which represents the name of the ship, if one is present. If the data type is equal to "nil" than there is no ship. And if the data type is equal to "-1", than the peek was invalid.

5.2 "Block" Interface

Prototype: (blockxy x-co y-co)

Description: The function take's two arguments. A "x-co" represents the x coordinate of the game board plain. And "y-co" represents the y coordinate of the game board plain.

Returns: The function returns a data type. If the data type variable is equal to "t", than the desired location was successfully blocked. And if its equal to "-1", then the block was invalid.

5.3 "Shoot" Interface

Prototype: (shoot x-co y-co)

Description: The function take's two argument. A "x-co" represents the x coordinate of the game board plain. An "y-co" represents the y coordinate of the game board plain.

Returns: The function returns a data type which contains data which states the percent destroyed of a ship, the type of ship, or "-1" represents a invalid shot.

6. Testing

By using Lisp we will be able to build this project using the “bottom-up” design methodology. We will build our code in blocks and test each component separately. Before the tournament game play begins we will pit our agent against itself and take notes on it's performance. This performance metric could be used to establish a baseline for comparison in later tests against other agents. There are also opportunities to test our agent in simulated tournament runs where we should be able to monitor it's performance against a variety of other agents. By comparing it's performance in this simulated tournaments to the original metric we should be able to draw some conclusions about what aspects we need to dedicate our attention to in the time remaining before the tournament. Since a large portion of our project is using probability and Bayes nets we may have to adjust the probability calculations to more accurately reflect tournament play.

7. Time line

Released:................................**Tue. Sept. 28**
Design complete:........................ **Thu. Nov. 4**
Implementation test 1:.................. **Tue. Nov. 16**
Implementation test 2:.................. **Tue. Dec. 7**
Debug and fine tuning:.................. **Nov. 16 – Dec. 7**
Tournament:............................. **Tue. Dec. 14**
Completion:............................. **Tue. Dec. 21**