**Project 4, Hashtables**

**Due: Please check due date on BlackBoard**

**Objectives**

To get experience creating and using a hashtable, and creating hash functions.

**Introduction**

Several artificial intelligence (AI) approaches for computer game playing involving creating dictionaries of possible game piece configurations and the moves which worked best for each configuration. Because the number of possible configurations is large (and some configurations are unlikely), it is not practical to create one large table of all the possible configurations. Instead, a common technique is to use a hash table to store game configurations which have been previously seen along with their likely outcomes. In this project, you will implement a simple tic-tac-toe game, using a hash table to store game configurations to play smarter. Your base program will be automatic play between two computer-controlled players -- a "smart" player that learns and a "dumb" player that chooses moves randomly.

**Application Setup**

You must have the following files, and they must obey the javadoc exactly. Note that there are several requirements stated in the file headers that MUST be followed. However, feel free to add other methods. The specified methods are not enough to solve this project.

1. [Hashtable.java](http://www.csee.umbc.edu/courses/undergraduate/341/spring14/projects/proj4/Hashtable.html)
2. [RandomAI.java](http://www.csee.umbc.edu/courses/undergraduate/341/spring14/projects/proj4/RandomAI.html)
3. [SmartPlayer.java](http://www.csee.umbc.edu/courses/undergraduate/341/spring14/projects/proj4/SmartPlayer.html)
4. [TicTacToe.java](http://www.csee.umbc.edu/courses/undergraduate/341/spring14/projects/proj4/TicTacToe.html)

Project4:  
  
Your Project4.java will have a method called play, which will take two Player objects and an int called numGames. It will play the two player objects against each other numGames number of times. After the game, play will print a final summary of the series of games and operations of the the tic-tac-toe program must be printed when your program terminates. The summary must contain the following information: statistics about the hash table: number of slots, number of entries, and number of collisions. statistics about wins and losses of the players advice about the best initial move for the smart player and outcomes resulting from that move (backed up with statistics from games played). The initial move suggestion must be based on statistics aquired during the game. Example final summary output. This sample is provided to illustrate all required data. The data does not necessarily reflect expected results. Your format may vary provided it is readable and contains all required data. Sample run:

run:

[java] FINAL REPORT:

[java]

[java] The number of slots is: 267

[java] The number of entries is: 218

[java] The % full is: 81.64

[java] The number of collisions is: 1087

[java] Smart player has won 342 times which is 34 percent

[java] Random has won 567 times which is 56 percent

[java] My favorite first move is:

[java] ...

[java] .OX

[java] ... [

java] Won 48.0 out of 71.0 which is 67.60%

**Requirements and Hints**

1. You may not use any built in hashCode() function to generate the hashcode in your TicTacToe class.
2. Everything should be in a package named proj4
3. All Hashtable operations should be O(1), regardless of load.
4. Your resulting player may be bad at TicTacToe; it may be worse than random. That is still acceptable, as long as all methods are within specification.

Class RandomAI

java.lang.Object

proj4.RandomAI

public class RandomAI

extends java.lang.Object

An AI that acts randomly. If it can, it will make some legal move on the board it is given.

Constructor Summary

Constructors

Constructor and Description

RandomAI()

Method Summary

Methods

Modifier and Type Method and Description

void move(TicTacToe t)

Takes a given board and makes a random move.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

RandomAI

public RandomAI()

Method Detail

move

public void move(TicTacToe t)

Takes a given board and makes a random move. If the board is full does nothing.

Parameters:

t - The board we are moving on.Class Hashtable<K,V>

java.lang.Object

proj4.Hashtable<K,V>

Type Parameters:

K - the type of the key.

V - values stored.

public class Hashtable<K,V>

extends java.lang.Object

A generic hashtable class, mapping keys to values. Uses the built in hashCode method on the keys. A generic hashtable class, mapping keys to values. Uses the built in hashCode method on the keys. Insert and get should be on average O(1), no matter what the user does. The hashtable should have a reasonable amount of memory usage; it should never have more than five times the number of slots as it has enteries. The hashtable should handle collisions gracefully in whatever way you choose. The number of collisions should be low relative to the size of the hashtable.

Constructor Summary

Constructors

Constructor and Description

Hashtable()

Method Summary

Methods

Modifier and Type Method and Description

boolean containsKey(K key)

Returns true if the key is contained in the table, false otherwise.

V get(K key)

Retrieves the object associated with this key.

int getPositionInHashtable(K key)

This method should tell you where in the hashtable this key will be stored, ASSUMING NO COLLISIONS.

int numCollisions()

Returns the number of times a collision has occurred.

int numEntries()

Returns the number of things actually stored in this hashtable.

int numSlots()

Returns the size of the array used to back your hashtable.

void put(K key, V value)

Puts a key value pair in the hashtable.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

Hashtable

public Hashtable()

Method Detail

get

public V get(K key)

Retrieves the object associated with this key. If the key maps to nothing, it should return null.

Parameters:

key - Key we're using for retrieval

Returns:

Associated object.

containsKey

public boolean containsKey(K key)

Returns true if the key is contained in the table, false otherwise.

Parameters:

key - Key we're looking for

Returns:

true if key maps to a value.

put

public void put(K key,

V value)

Puts a key value pair in the hashtable. If this key already corresponds to a value, that value is overwritten.

Parameters:

key - The key used for retreival later

value - Corresponding value.

numSlots

public int numSlots()

Returns the size of the array used to back your hashtable. The array slots need not be full to count.

Returns:

Size of the hashtable array.

numEntries

public int numEntries()

Returns the number of things actually stored in this hashtable.

Returns:

Number of elements stored.

numCollisions

public int numCollisions()

Returns the number of times a collision has occurred. If you are handling collisions using by storing multiple entries on one cell, you may return the number of doubled up entries. If you are handling collisions by rehashing (using a secondary hash function when collisions are detected) have an instance variable that keeps track of how often this has happened.

Returns:

Number of collisions.

getPositionInHashtable

public int getPositionInHashtable(K key)

This method should tell you where in the hashtable this key will be stored, ASSUMING NO COLLISIONS. Depending on how you design your project, this may be as simple as key.hashCode() % array.length.

Parameters:

key - The key we're trying to figure out where to store

Returns:

The index in the array we would store this key.

Class SmartPlayer

java.lang.Object

proj4.SmartPlayer

public class SmartPlayer

extends java.lang.Object

A smart(er) TicTacToe player. It keeps track of moves it has already made, and when it can tries take the one that has resulted in the most wins. Failing that, it will take an action that has resulted in the least losses. If all actions are equal, it will choose one at random. This object must be told when a new game has started (and what player it is playing) and also it must see the game when it is finished (otherwise there is no way for it to know if an option resulted in a win or a loss. This class must have a Hashtable that maps TicTacToe boards to statistics about the results of the game

Constructor Summary

Constructors

Constructor and Description

SmartPlayer()

Method Summary

Methods

Modifier and Type Method and Description

void endGame(TicTacToe finalBoard)

Let's smartAI know that the game has ended.

TicTacToe[] getSuccessors(TicTacToe t)

This should return every board we've played that came after the board t.

void move(TicTacToe t)

Makes a move based on past experince.

void newGame(int player)

int numberOfTimesSeen(TicTacToe t)

How many times we've played this particular board

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

SmartPlayer

public SmartPlayer()

Method Detail

newGame

public void newGame(int player)

move

public void move(TicTacToe t)

Makes a move based on past experince. Tries take the one that has resulted in the most wins. Failing that, it will take an action that has resulted in the least losses. If all actions are equal, it will choose one at random.

Parameters:

t -

endGame

public void endGame(TicTacToe finalBoard)

Let's smartAI know that the game has ended. This method will probably require going back through all the moves in this game so far and updating them with whether they ended in a win, a loss, or a draw.

Parameters:

finalBoard -

numberOfTimesSeen

public int numberOfTimesSeen(TicTacToe t)

How many times we've played this particular board

Parameters:

The - board we're looking at

Returns:

Number of times smartAI has played on it.

getSuccessors

public TicTacToe[] getSuccessors(TicTacToe t)

This should return every board we've played that came after the board t.

Parameters:

t - The current board.

Returns:

Every board we've experinced that is one move after t.

Class TicTacToe

java.lang.Object

proj4.TicTacToe

public class TicTacToe

extends java.lang.Object

A very simple class representing a game of TicTacToe. The first player (X) is player 1, the second player (O) is player 2. Moves are indexed starting at zero. You may not call the built in hashCode() function at any point in this file.

Constructor Summary

Constructors

Constructor and Description

TicTacToe()

Method Summary

Methods

Modifier and Type Method and Description

int getWinner()

Tells you who won the game.

int hashCode()

Returns a hashcode for this object.

boolean isOver()

Test if the game is over

boolean move(int row, int col)

Makes a move in the given cell if it is empty (returns false if this is not possible).

int playerAt(int row, int col)

Returns the current move made at the given location

java.lang.String toString()

String representation of this object.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, notify, notifyAll, wait, wait, wait

Constructor Detail

TicTacToe

public TicTacToe()

Method Detail

move

public boolean move(int row,

int col)

Makes a move in the given cell if it is empty (returns false if this is not possible). Cell indexing starts at 0.

Parameters:

row - The row of the move

col - The column of the move

Returns:

Whether or not the move was made successfully.

getWinner

public int getWinner()

Tells you who won the game.

Returns: Returns 1 if player 1 won, 2 if player 2 won, and 0 if it is a draw. Should return -1 if the game is not over.

isOver: public boolean isOver() Test if the game is over

Returns: True if the game has ended, false otherwise.

playerAt: public int playerAt(int row, int col)

Returns the current move made at the given location

Parameters: row - The row; col - The column

Returns: A 1, 2, or a 0 based on the player's move.

toString: public java.lang.String toString() - String representation of this object.

Returns: Object in string form. Should be -'s for spaces, X and O for the player moves. For example: XO- XXO ---

hashCode: public int hashCode()

Returns a hashcode for this object. Code this however you like, as long as you don't use the hashCode method for anything else in java. Your hashtable will use this to find slots for the object.

Returns: hash code