TicTacToe v1.0

Generated by Doxygen 1.12.0

1 liciacioe	1
1.1 TIC TAC TOE - CSC1103 & CSC1104 Project	 1
1.1.1 Installation Instructions (Linux)	 1
1.1.2 Installation Instructions (MacOS)	 2
1.1.3 Installation Instructions (Windows)	 3
1.1.4 BASIC REQUIREMENTS (BOTH)	 4
1.1.5 [^1]PM-CSC1103 REQUIRMENTS	 4
1.1.6 [^2]COA-CSC1104 REQUIRMENTS	 5
2 Doxygen Documentation Generation and Viewing Instructions	5
2.1 Steps to Generate and View the Documentation:	 5
2.1.1 1. Run the Doxygen Configuration File:	 5
2.1.2 2. View the HTML Documentation:	 5
2.1.3 3. Generate and View LaTeX Documentation:	 5
2.2 Troubleshooting:	 6
3 Data Structure Index	6
3.1 Data Structures	 6
4 File Index	6
4.1 File List	 6
5 Data Structure Documentation	7
5.1 BoardState Struct Reference	 7
5.1.1 Detailed Description	 7
5.1.2 Field Documentation	 7
5.2 BtnPos Struct Reference	 7
5.2.1 Detailed Description	 7
5.2.2 Field Documentation	 8
5.3 Dataset Struct Reference	 8
5.3.1 Detailed Description	 8
5.3.2 Field Documentation	 8
5.4 PlayerMode Struct Reference	 8
5.4.1 Detailed Description	 8
5.4.2 Field Documentation	 9
5.5 Position Struct Reference	 9
5.5.1 Detailed Description	 9
5.5.2 Field Documentation	 9
6 File Documentation	9
6.1 Documentation/README.md File Reference	 9
6.2 header/elapsedTime.h File Reference	 9
6.2.1 Detailed Description	 10
6.2.2 Function Documentation	 10

6.3 elapsedTime.h	 11
6.4 header/importData.h File Reference	 11
6.4.1 Detailed Description	 11
6.4.2 Macro Definition Documentation	 12
6.4.3 Function Documentation	 12
6.5 importData.h	 14
6.6 header/macros.h File Reference	 14
6.6.1 Detailed Description	 15
6.6.2 Macro Definition Documentation	 15
6.7 macros.h	 17
6.8 header/main.h File Reference	 18
6.8.1 Detailed Description	 19
6.8.2 Function Documentation	 19
6.9 main.h	 22
6.10 header/minimax.h File Reference	 22
6.10.1 Detailed Description	 23
6.10.2 Macro Definition Documentation	 23
6.10.3 Function Documentation	 23
6.11 minimax.h	 28
6.12 header/ml-naive-bayes.h File Reference	 28
6.12.1 Detailed Description	 29
6.12.2 Macro Definition Documentation	 30
6.12.3 Function Documentation	 30
6.13 ml-naive-bayes.h	 34
6.14 mainpage.md File Reference	 34
6.15 src/elapsedTime.c File Reference	 34
6.15.1 Function Documentation	 35
6.15.2 Variable Documentation	 35
6.16 elapsedTime.c	 35
6.17 src/importData.c File Reference	 36
6.17.1 Function Documentation	 36
6.17.2 Variable Documentation	 38
6.18 importData.c	 39
6.19 src/main.c File Reference	 40
6.19.1 Function Documentation	 41
6.19.2 Variable Documentation	 44
6.20 main.c	 46
6.21 src/minimax.c File Reference	 50
6.21.1 Function Documentation	 51
6.21.2 Variable Documentation	 56
6.22 minimax.c	 56
6.23 src/ml-naive-bayes.c File Reference	 61

1 TicTacToe 1

ln	ndex	75
	6.24 ml-naive-bayes.c	68
	6.23.2 Variable Documentation	66
	6.23.1 Function Documentation	62

1 TicTacToe

1.1 TIC TAC TOE - CSC1103 & CSC1104 Project

1.1.1 Installation Instructions (Linux)

To get started with the project, ensure you have the following installed:

Important

Ensure that you follow all the instruction in the link below!

1. Setup Docker's apt repository

```
# Add Docker's official GPG key:
sudo apt-get update
sudo apt-get install ca-certificates curl
sudo install -m 0755 -d /etc/apt/keyrings
sudo curl -fsSL https://download.docker.com/linux/debian/gpg -o /etc/apt/keyrings/docker.asc
sudo chmod a+r /etc/apt/keyrings/docker.asc
# Add the repository to Apt sources:
echo \
   "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc]
https://download.docker.com/linux/debian \
   $(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt-get update
```

2. Installing xhost(X11)

```
sudo apt-get install x11-apps
sudo apt-get install x11-xserver-utils
```

1. Installing Docker Desktop

sudo apt-get install docker-ce docker-ce-cli containerd io docker-buildx-plugin docker-compose-plugin

Remarks

Adding this command in startup.sh will make your life easier.

```
systemctl --user start docker-desktop
```

1.1.1.1 Building the Project via Docker (Linux)

After setting up WSL2 and Docker, you can choose to either load a Docker image or build the Docker image yourself.

1.1.1.1.1 [OPTIONAL] Loading Docker Image

```
sudo docker load -i FILE_NAME.tar
```

1.1.1.1.2 Build and Run TicTacToe Application on Docker (Linux)

sudo ./run_docker.sh

Warning

Warning for Docker Building:

If compile.sh or run_docker.sh is not found and it's clearly in the directory, run the following command to convert it to Unix line endings:

dos2unix SCRIPT_NAME.sh

Remarks

Saving Docker Image

./run_script.sh -s

1.1.2 Installation Instructions (MacOS)

To get started with the project, ensure you have the following installed:

- 1. **Docker Desktop** Select based on your specs.
 - Install Docker Desktop
 - · Select "Download for Mac Intel Chip"
 - · Select "Download for Mac Apple Silicon"

2. Installation Docker Desktop

- · Go to "Downloads" and double click "Docker.dmg"
- Drag and Drop "Docker" to "Application"
- "Launch Pad" either search or swipe to find "Docker"

3. Verify Docker Desktop

- Launch the application and select "Recommended Settings" OR
- Type the command in terminal "docker --version"
- 4. **Verify Docker Image and Containers** Now create a new directory/folder and "cd" to that folder. inside terminal use the following command and wait for "Download complete"
 - · docker pull hello-world
 - · Verify docker image in "Docker Desktop"

Under the action tab click on "Run" OR From terminal type the following.

docker run hello-world

To verify container. Switch to the containers tab. Under the name tab and click on the "highlighted blue" and view the logs. OR Use the following command.

docker ps -a

1.1.2.1 Run TicTacToe Application on Docker (MacOS)

Important

- **Pre-requisite application to be install.
- **BEFORE BUILDING DOCKER IMAGE OR CONTAINER.
- 1. Install brew
 - Install brew
- 2. install xquartz from terminal brew install --cask xquartz
- 3. xquarts settings
 - setting -> security -> allow connections from network clients
 - · Restart xquartz
- 4. Terminal command xhost + 127.0.0.1
- 5. Build and Run Docker docker-compose up --build -d

1.1.3 Installation Instructions (Windows)

To get started with the project, ensure you have the following installed:

- 1. Docker Desktop
 - Install Docker Desktop
- 2. WSL2 (Windows Subsystem for Linux)
 - Follow Microsoft's guide to install WSL2
 - Install Ubuntu with the following command: wsl --install -d Ubuntu
 - Set Ubuntu as your default WSL distribution: wsl --set-default Ubuntu
 - Check default: wsl -l -v

1.1.3.1 Building the Project via Docker (Windows)

After setting up WSL2 and Docker, you can choose to either load a Docker image or build the Docker image yourself.

1.1.3.1.1 [OPTIONAL] Loading Docker Image

docker load -i FILE_NAME.tar

1	1 4	2 1	2	Ruild	and Run	TicTacTop	Application	on Docker	(Windows)
Ι.		J. I		Dullu	aliu nuli	TIC TACTOR	ADDIICALION	OII DOCKEI	(WIIIGOWS

./run_docker.sh

Warning

Warning for Docker Building: If <code>compile.sh</code> or <code>run_docker.sh</code> is not found and it's clearly in the directory, run the following command to convert it to Unix line endings: $dos2unix SCRIPT_NAME.sh$

Remarks

Saving Docker Image

./run_script.sh -s

1.1.3.2 Build and Run the Project in Windows (w/o Docker)

Attention

Not recommended, unless you know what you're doing and install the right packages. (refer Dockerfile)

1.1.4	BASIC F	REQUIREMENTS	(BOTH)
-------	---------	--------------	--------

√	GUI (GTK)
√	2 Player Mode
√	1 Player Mode ("Perfect" Minimax)
<u> </u>	Winning Logic
✓	GUI indication when player WIN (e.g, blinking))
1.1.5	5 [^1]PM-CSC1103 REQUIRMENTS
√	Improve Minimax memory usage
✓	Implement ML Algorithm (80:20)
V	Implement ML Algorithm (80:20) [TIP] Linear regression, Navie bayes, Neural network and Reinforcement learning

1.1.6 [^2]COA-CSC1104 REQUIRMENTS

☑ Replace one function with assembly

[TIP] Use inline assembly code in C source file, or linking separate C and assembly object files.

[^1]: PROGRAMMING METHODOLOGY. [^2]: COMPUTER ORGANIZATION AND ARCH.

2 Doxygen Documentation Generation and Viewing Instructions

This guide provides instructions for generating, extracting, and viewing the Doxygen documentation in HTML and LaTeX formats.

2.1 Steps to Generate and View the Documentation:

2.1.1 1. Run the Doxygen Configuration File:

- Locate the Doxyfile in the project directory.
- Run Doxygen using the following command in your terminal: bash doxygen Doxyfile
- This will generate two folders in ./docs:
 - html (for HTML documentation)
 - latex (for LaTeX documentation)

2.1.2 2. View the HTML Documentation:

- · Navigate to the html folder generated by Doxygen.
- Locate the file named index.html.
- Open index.html in your preferred web browser:
 - Double-click the file to open it using the default web browser.
 - Alternatively, right-click the file and select Open With to choose a specific browser (e.g., Chrome, Firefox, Edge).

2.1.2.1 Recommended Browsers:

- Google Chrome (latest version)
- · Mozilla Firefox (latest version)
- · Microsoft Edge (latest version)
- · Safari (for macOS users)

2.1.3 3. Generate and View LaTeX Documentation:

- Navigate to the latex folder generated by Doxygen.
- · Compile the LaTeX files into a PDF:
 - Ensure you have a LaTeX distribution installed, such as TeX Live, MiKTeX, or MacTeX.
 - Run the following commands in the terminal: bash cd latex make
 - This will produce a PDF file (e.g., refman.pdf) in the latex folder.
- · Open the PDF using a PDF viewer like Adobe Acrobat Reader or any other suitable application.

2.2 Troubleshooting:

- Doxygen Command Not Found:
 - Ensure Doxygen is installed and added to your system's PATH.
 - Download Doxygen from https://www.doxygen.nl/download.html if not already installed.
- LaTeX Make Errors:
 - Ensure a LaTeX distribution is installed and properly set up.
 - Check the make command output for details on missing dependencies.

3 Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

BoardState Stores the current state of the Tic-Tac-Toe board along with the best move	7
BtnPos	
Stores the position of a button in the game grid	7
Dataset	
Structure to hold a Tic-Tac-Toe board state and its outcome	8
PlayerMode	
Stores the current game mode and its textual representation	8
Position	
Represents a position on the Tic-Tac-Toe grid	g

4 File Index

4.1 File List

Here is a list of all files with brief descriptions:

header/elapsedTime.h Provides functionality for measuring elapsed time for profiling purposes	9
header/importData.h Header file for handling dataset import and manipulation for Tic-Tac-Toe game data	11
header/macros.h Header file containing macros, constants, and structure definitions for the Tic-Tac-Toe game	14
header/main.h Header file for the Tic-Tac-Toe game logic	18
header/minimax.h Header file for the Tic-Tac-Toe Minimax algorithm	22
header/ml-naive-bayes.h Header file for Naive Bayes classifier functions for Tic-Tac-Toe outcome prediction	28
src/elapsedTime.c	34

src/importData.c	36
src/main.c	40
src/minimax.c	50
src/ml-naive-bayes.c	61

5 Data Structure Documentation

5.1 BoardState Struct Reference

Stores the current state of the Tic-Tac-Toe board along with the best move.

#include <minimax.h>

Data Fields

- int board [3][3]
- struct Position bestMove

5.1.1 Detailed Description

Stores the current state of the Tic-Tac-Toe board along with the best move.

< Include macro definitions

This structure holds a 3x3 board array and the best move associated with that board state. It is used for storing and comparing previous board states and their corresponding optimal moves.

Definition at line 29 of file minimax.h.

5.1.2 Field Documentation

bestMove

```
struct Position bestMove
The best move for the bot
Definition at line 32 of file minimax.h.
```

board

```
int board[3][3]
```

The Tic-Tac-Toe board

Definition at line 31 of file minimax.h.

The documentation for this struct was generated from the following file:

header/minimax.h

5.2 BtnPos Struct Reference

Stores the position of a button in the game grid.

#include <main.h>

Data Fields

• int pos [2]

5.2.1 Detailed Description

Stores the position of a button in the game grid.

This structure contains an array pos[2] that holds the row and column of the button in the grid. Definition at line 80 of file main.h.

5.2.2 Field Documentation

pos

```
int pos[2]
```

Array to hold row and column

Definition at line 82 of file main.h.

The documentation for this struct was generated from the following file:

header/main.h

5.3 Dataset Struct Reference

Structure to hold a Tic-Tac-Toe board state and its outcome.

```
#include <importData.h>
```

Data Fields

- char grid [3][3]
- char outcome [9]

5.3.1 Detailed Description

Structure to hold a Tic-Tac-Toe board state and its outcome.

This structure represents the board state as a 3x3 grid and the outcome as a string. Definition at line 28 of file importData.h.

5.3.2 Field Documentation

grid

```
char grid[3][3]
```

3x3 grid representing the board state

Definition at line 30 of file importData.h.

outcome

```
char outcome[9]
```

Outcome of the board state

Definition at line 31 of file importData.h.

The documentation for this struct was generated from the following file:

· header/importData.h

5.4 PlayerMode Struct Reference

Stores the current game mode and its textual representation.

```
#include <main.h>
```

Data Fields

- char txt [2]
- int mode

5.4.1 Detailed Description

Stores the current game mode and its textual representation.

This structure holds the mode of the game (e.g., player vs player, player vs bot) and a textual representation of the mode for display.

Definition at line 92 of file main.h.

5.4.2 Field Documentation

mode

int mode

Integer value representing the current game mode Definition at line 95 of file main.h.

txt

```
char txt[2]
```

Textual representation of the current game mode

Definition at line 94 of file main.h.

The documentation for this struct was generated from the following file:

· header/main.h

5.5 Position Struct Reference

Represents a position on the Tic-Tac-Toe grid.

```
#include <macros.h>
```

Data Fields

- int row
- int col

5.5.1 Detailed Description

Represents a position on the Tic-Tac-Toe grid. Definition at line 65 of file macros.h.

5.5.2 Field Documentation

col

int col

Column index (0-2)

Definition at line 68 of file macros.h.

row

int row

Row index (0-2)

Definition at line 67 of file macros.h.

The documentation for this struct was generated from the following file:

· header/macros.h

6 File Documentation

6.1 Documentation/README.md File Reference

6.2 header/elapsedTime.h File Reference

Provides functionality for measuring elapsed time for profiling purposes.

```
#include <macros.h>
#include <sys/time.h>
```

Functions

void startElapseTime ()

Starts the elapsed time tracking.

void stopElapseTime (char *str)

Stops the elapsed time tracking and outputs the result.

6.2.1 Detailed Description

Provides functionality for measuring elapsed time for profiling purposes.

Author

jacktan-jk

Version

1.0

Date

2024-11-13

Copyright

Copyright (c) 2024

This header file declares functions and macros to track the start and stop times of operations, enabling performance measurement. The code is conditionally compiled based on the <code>DISABLE_ELAPSED</code> macro, allowing profiling code to be included or excluded as needed.

Definition in file elapsedTime.h.

6.2.2 Function Documentation

startElapseTime()

```
void startElapseTime ()
```

Starts the elapsed time tracking.

Captures the current time and stores it in gStartTime to mark the beginning of an elapsed time measurement. Only operates if DISABLE_ELAPSED is not defined, allowing conditional compilation for performance tracking. Definition at line 9 of file elapsedTime.c.

stopElapseTime()

Stops the elapsed time tracking and outputs the result.

 $\textbf{Calculates the time elapsed since} \ \texttt{startElapseTime} \ \textbf{and outputs it in seconds using the provided label}.$

Parameters

str Label describing the operation or section being timed.

Only operates if <code>DISABLE_ELAPSED</code> is not defined, and outputs timing information through <code>PRINT_DEBUG</code> for profiling and debugging.

Definition at line 17 of file elapsedTime.c.

6.3 elapsedTime.h

6.3 elapsedTime.h

Go to the documentation of this file.

```
00001
00016 #ifndef ELAPSED_TIME_H
00017 #define ELAPSED_TIME_H
00018
00019 #include <macros.h>
00020 #include <sys/time.h>
00021
00031 void startElapseTime();
00032
00044 void stopElapseTime(char *str);
00045
00045
00046 #endif // ELAPSED_TIME_H
```

6.4 header/importData.h File Reference

Header file for handling dataset import and manipulation for Tic-Tac-Toe game data.

```
#include <macros.h>
```

Data Structures

struct Dataset

Structure to hold a Tic-Tac-Toe board state and its outcome.

Macros

- #define RES_PATH "./resources/"
- #define DATA_PATH "tic-tac-toe.data"
- #define TRAIN PATH "training-"
- #define TEST_PATH "testing-"

Functions

• int readDataset (const char *filename, bool split)

Reads a dataset from a file and optionally randomizes entries for training and testing.

• int splitFile ()

Splits the dataset into training and testing files with an 80-20 ratio.

void getRandomNo (int random[DATA_SIZE])

Generates an array of unique random integers within the range of the dataset size.

int getTrainingData (struct Dataset **d)

Retrieves the training data from a file and returns its length.

int getTestingData (struct Dataset **d)

Retrieves the testing data from a file and returns its length.

6.4.1 Detailed Description

Header file for handling dataset import and manipulation for Tic-Tac-Toe game data.

Author

jacktan-jk

Version

1.0

Date

2024-11-12

This file contains function declarations and structures to read, split, and manage Tic-Tac-Toe game data used for training and testing.

Definition in file importData.h.

6.4.2 Macro Definition Documentation

DATA_PATH

```
#define DATA_PATH "tic-tac-toe.data" Name of the primary dataset file Definition at line 18 of file importData.h.
```

RES_PATH

```
#define RES_PATH "./resources/"
Path to resources directory
Definition at line 17 of file importData.h.
```

TEST PATH

```
#define TEST_PATH "testing-"
Prefix for testing data file
Definition at line 20 of file importData.h.
```

TRAIN PATH

```
#define TRAIN_PATH "training-"
Prefix for training data file
Definition at line 19 of file importData.h.
```

6.4.3 Function Documentation

getRandomNo()

Generates an array of unique random integers within the range of the dataset size.

This function populates an array with unique random integers between 0 and DATA_SIZE - 1. It ensures that each integer appears only once by checking a check array to track used indices. This can be used for randomizing the order of data for splitting purposes.

Parameters

random Array to store the generated unique random integers.

See also

DATA SIZE

Definition at line 143 of file importData.c.

getTestingData()

Retrieves the testing data from a file and returns its length.

This function zeroes out the data array for the length of the testing set, reads the dataset from the specified testingFile, and assigns the data pointer to the provided dataset pointer. Returns the length of the testing data loaded.

Parameters

d Pointer to a dataset pointer that will reference the loaded testing data array.

Returns

The number of testing entries loaded (i.e., len_test).

See also

```
readDataset, testingFile
```

Definition at line 175 of file importData.c.

getTrainingData()

Retrieves the training data from a file and returns its length.

This function initializes the data array to zero for the length of the training set, reads the dataset from the specified trainingFile, and assigns the data pointer to the provided dataset pointer. Returns the length of the training data loaded.

Parameters

d Pointer to a dataset pointer that will reference the loaded training data array.

Returns

The number of training entries loaded (i.e., len_train).

See also

readDataset, trainingFile

Definition at line 167 of file importData.c.

readDataset()

Reads a dataset from a file and optionally randomizes entries for training and testing.

Opens a file to read each line as a Tic Tac Toe board state, populating a grid structure where 'x', 'o', and 'b' represent the Bot, Player 1, and empty cells, respectively. Each board state is followed by an outcome that is stored within the dataset. If split is true, entries are randomized using an array of unique indices for shuffling.

Parameters

filename	The name of the dataset file to read.
split	Boolean indicating whether to randomize entries for dataset splitting.

Returns

int SUCCESS (0) if reading is successful, BAD_PARAM (-5) if the file cannot be opened, or the return value of splitFile() if split is enabled.

See also

getRandomNo, splitFile

Definition at line 44 of file importData.c.

splitFile()

```
int splitFile ()
```

Splits the dataset into training and testing files with an 80-20 ratio.

This function separates the dataset into two parts: 80% for training and 20% for testing. The training portion is written to trainingFile, and the testing portion is written to testingFile. Each entry consists of a 3x3 grid representing the Tic Tac Toe board and the outcome of that board.

Returns

int SUCCESS (0) if both files are written successfully, BAD PARAM (-5) if either file cannot be opened.

See also

```
data, trainingFile, testingFile
```

Definition at line 90 of file importData.c.

6.5 importData.h

Go to the documentation of this file.

```
00011 #ifndef IMPORTDATA H
00012 #define IMPORTDATA_H
00013
00014 #include <macros.h>
00015
00016 // changed to 100 for testing. make sure to chg back
00017 #define RES_PATH "./resources/"
00018 #define DATA_PATH "tic-tac-toe.data"
00019 #define TRAIN_PATH "training-"
00020 #define TEST_PATH "testing-
00028 struct Dataset
00029 {
00030
           char grid[3][3];
00031
          char outcome[9];
00032 };
00033
00049 int readDataset(const char *filename, bool split);
00064 int splitFile();
00065
00077 void getRandomNo(int random[DATA_SIZE]);
00078
00092 int getTrainingData(struct Dataset **d);
00107 int getTestingData(struct Dataset **d);
00108
00109 #endif // IMPORTDATA H
```

6.6 header/macros.h File Reference

Header file containing macros, constants, and structure definitions for the Tic-Tac-Toe game.

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <time.h>
#include <string.h>
```

Data Structures

struct Position

Represents a position on the Tic-Tac-Toe grid.

Macros

- #define PLAY 0x69
- #define TIE 0xFF

- #define WIN 0xAA
- #define SUCCESS 0
- #define ERROR -1
- #define BAD PARAM -5
- #define MODE 2P 0
- #define MODE MM 1
- #define MODE ML 2
- #define EMPTY 0
- #define PLAYER1 1
- #define BOT 2
- #define DATA_SIZE 958
- #define CLASSES 2
- #define DEBUG 1
- #define MINIMAX_GODMODE
- #define DISABLE LOOKUP
- #define DISABLE_ELAPSED
- #define DISABLE_ASM
- #define PRINT_DEBUG(...)

6.6.1 Detailed Description

Header file containing macros, constants, and structure definitions for the Tic-Tac-Toe game.

Author

jacktan-jk

Version

1.0

Date

2024-11-13

Copyright

Copyright (c) 2024

This file defines various constants for game modes, player identifiers, error codes, and debugging options. It also includes the Position structure for grid positions in the game.

Definition in file macros.h.

6.6.2 Macro Definition Documentation

BAD PARAM

#define BAD_PARAM -5

Bad parameter error

Definition at line 30 of file macros.h.

BOT

#define BOT 2

Bot (Player 2)

Definition at line 40 of file macros.h.

CLASSES

#define CLASSES 2

Number of outcome classes

Definition at line 44 of file macros.h.

DATA_SIZE

#define DATA_SIZE 958

Dataset size

Definition at line 43 of file macros.h.

DEBUG

#define DEBUG 1

Enable debug messages

Definition at line 47 of file macros.h.

DISABLE_ASM

#define DISABLE_ASM

Disable ASM functions

Definition at line 51 of file macros.h.

DISABLE_ELAPSED

#define DISABLE_ELAPSED

Disable Elapsed time function

Definition at line 50 of file macros.h.

DISABLE_LOOKUP

#define DISABLE_LOOKUP

Disable Minimax lookup table

Definition at line 49 of file macros.h.

EMPTY

#define EMPTY 0

Empty cell

Definition at line 38 of file macros.h.

ERROR

#define ERROR -1

Error

Definition at line 29 of file macros.h.

MINIMAX_GODMODE

#define MINIMAX_GODMODE

Minimax god mode toggle

Definition at line 48 of file macros.h.

MODE_2P

#define MODE_2P 0

Two-player mode

Definition at line 33 of file macros.h.

MODE ML

#define MODE_ML 2

Machine Learning mode

Definition at line 35 of file macros.h.

6.7 macros.h 17

MODE_MM

```
#define MODE_MM 1
Minimax mode
Definition at line 34 of file macros.h.
```

PLAY

```
#define PLAY 0x69

Player move

Definition at line 25 of file macros.h.
```

PLAYER1

```
#define PLAYER1 1
Player 1
Definition at line 39 of file macros.h.
```

PRINT_DEBUG

SUCCESS

```
#define SUCCESS 0
Success
Definition at line 28 of file macros.h.
```

TIE

```
#define TIE 0xFF
Tie state
Definition at line 26 of file macros.h.
```

WIN

```
#define WIN 0xAA
Winning state
Definition at line 27 of file macros.h.
```

6.7 macros.h

Go to the documentation of this file.

```
00001
00015 #ifndef MACROS_H
00016 #define MACROS_H
00017
00018 #include <stdio.h>
00019 #include <stdib.h>
00020 #include <stdbool.h>
00021 #include <time.h>
00022 #include <string.h>
00022 #include <string.h>
00022 #oostants for game states and player identifiers
00025 #define PLAY 0x69
00026 #define TIE 0xFF
00027 #define WIN 0xAA
00028 #define SUCCESS 0
00029 #define ERROR -1
00030 #define BAD_PARAM -5
00032 // Game modes
00033 #define MODE_2P 0
00034 #define MODE_MM 1
```

```
00035 #define MODE_ML 2
00037 // Player identifiers
00038 #define EMPTY 0
00039 #define PLAYER1 1
00040 #define BOT 2
00042 // Data constants
00043 #define DATA_SIZE 958
00044 #define CLASSES 2
00046 // Debugging and configuration options
00047 #define DEBUG 1
00048 #define MINIMAX_GODMODE 0
00049 #define DISABLE_LOOKUP 0
00050 #define DISABLE_ELAPSED 0
00051 #define DISABLE_ASM
00053 #if DEBUG
00054 #define PRINT_DEBUG(...) printf(__VA_ARGS__);
00055 #else
00056 #define PRINT_DEBUG(...)
00057 #endif
00058
00065 struct Position
00066 {
00067
          int row;
00068
          int col;
00069 };
00071 #endif // MACROS_H
```

6.8 header/main.h File Reference

Header file for the Tic-Tac-Toe game logic.

```
#include <gtk/gtk.h>
#include <macros.h>
#include <minimax.h>
#include <ml-naive-bayes.h>
#include <elapsedTime.h>
```

Data Structures

struct BtnPos

Stores the position of a button in the game grid.

struct PlayerMode

Stores the current game mode and its textual representation.

Functions

• int doBOTmove ()

Executes the bot's move based on the current game mode.

• int chkPlayerWin ()

Checks the current game board for a win or tie.

• void clearGrid ()

Clears the game board and resets the player's turn.

• void updateScoreBtn (gpointer data)

Updates the score display on the button.

void on btnGrid clicked (GtkWidget *widget, gpointer data)

Callback function for handling button clicks on the game grid.

void on_btnScore_clicked (GtkWidget *widget, gpointer data)

Handles button click for score.

• void showWin ()

Clears the winning positions and resets the grid.

6.8.1 Detailed Description

Header file for the Tic-Tac-Toe game logic.

Author

jacktan-jk

Version

1.0

Date

2024-11-12

Copyright

Copyright (c) 2024

This file contains function declarations, structure definitions, and constants for handling game logic, button interactions, and the bot's behavior.

Definition in file main.h.

6.8.2 Function Documentation

chkPlayerWin()

```
int chkPlayerWin ()
```

Checks the current game board for a win or tie.

This function checks all possible win conditions:

- · Diagonals
- Rows
- · Columns

If there is a winning line, it marks the winning positions and returns WIN. If there are no winning conditions and the board is full, it returns TIE. If there are unclicked positions left, it returns PLAY.

Returns

WIN if there is a winner, TIE if the game is a tie, PLAY if the game is still ongoing.

See also

iBoard, iWinPos

Definition at line 234 of file main.c.

clearGrid()

```
void clearGrid ()
```

Clears the game board and resets the player's turn.

This function is used to reset the game board for a new round. It clears the labels on the buttons in the grid and resets the internal board state (iBoard) to 0. It also sets the player turn back to player 1.

- Sets all button labels in the btnGrid to an empty string.
- Resets all values in the iBoard array to 0, indicating no moves.
- Resets isPlayer1Turn to true, indicating it's Player 1's turn.

See also

iBoard

btnGrid

isPlayer1Turn

Definition at line 29 of file main.c.

doBOTmove()

```
int doBOTmove ()
```

Executes the bot's move based on the current game mode.

In MM mode:

- · Performs a minimax move.
- 20% chance of the minimax randomly selects a position.

In ML mode, the bot uses machine learning to determine the best position.

The function also measures and logs the time taken for the minimax move.

Returns

SUCCESS if the bot's move was made successfully.

See also

```
playerMode, isMLAvail, iBoard, findBestMove, getBestPosition, btnGrid
```

Definition at line 181 of file main.c.

on_btnGrid_clicked()

Callback function for handling button clicks on the game grid.

This function handles the logic for a player's move when a button in the game grid is clicked. It updates the game state, checks for a winner or tie, and updates the score display. It also handles player turns, Bot moves (if applicable), and resets the game board when the game state changes.

Parameters

widget	The GtkWidget that was clicked (the button in the grid).
data	Additional data passed to the callback (usually the score display data).

- If the game state is not PLAY, the game will be reset, and the score updated.
- If the clicked button already has a label, the function returns early (no action is taken).
- If the clicked button is empty, the move is recorded in the iBoard array (Player 1 or MM or ML).
- After each move, the game checks for a win or tie condition using chkPlayerWin().
- If Player 1 or Player 2 wins, the score is updated, and the win condition is shown.
- If the game ends in a tie, the tie score is updated.
- If the game is in 2P mode, turns alternate between Player 1 and Player 2.
- In MM mode, the Minimax will automatically make a move after Player 1's turn.
- In **ML mode**, the dataset is re-read and initialized after the game ends.

See also

```
iBoard, isPlayer1Turn, iPlayer1_score, iPlayer2_score, iTie_score playerMode, updateScoreBtn, chkPlayerWin, doBOTmove, showWin PLAY, TIE, WIN
```

Definition at line 60 of file main.c.

on_btnScore_clicked()

Handles button click for score.

Toggles the player mode and updates the displayed score.

Parameters

widget	The widget that triggered the event.
data	Additional data passed to the callback.

See also

playerMode, isMLAvail, isPlayer1Turn, updateScoreBtn, clearGrid

Definition at line 131 of file main.c.

showWin()

```
void showWin ()
```

Clears the winning positions and resets the grid.

Iterates over the win positions and clears any displayed labels, resetting the grid to its initial state.

See also

iWinPos, btnGrid

Definition at line 159 of file main.c.

updateScoreBtn()

Updates the score display on the button.

This function updates the label on a score button to display the current scores for Player 1, Player 2, and Ties. It changes the text formatting depending on which player's turn it is, highlighting the active player.

Parameters

data	A gpointer (usually a button widget) that is used to update the label.
------	------------------------------------------------------------------------

- The function checks if it's Player 1's turn and updates the score display with a bold label for Player 1, or Player 2's turn with Player 2's score in bold.
- The button text is updated using gtk_button_set_label(), and the label markup is updated using gtk_label_set_markup().
- The score includes Player 1's score, Player 2's score, the tie count, and the current game mode (player← Mode.txt).

See also

iPlayer1_score, iTie_score, iPlayer2_score, playerMode

Definition at line 42 of file main.c.

6.9 main.h

Go to the documentation of this file.

```
00001
00014 #ifndef MAIN_H // Start of include guard
00015 #define MAIN H
00016 #include <gtk/gtk.h>
00018 #include <macros.h>
00019 #include <minimax.h>
00020 #include <ml-naive-bayes.h>
00021 #include <elapsedTime.h>
00022
00023 /*=======
00024 GLOBAL DECLARATION
00025 ===
00080 typedef struct
00081 {
00082
          int pos[2];
00083 } BtnPos;
00084
00092 struct PlayerMode
00093 {
00094
          char txt[2];
00095
          int mode:
00096 };
00097
00112 int doBOTmove();
00113
00129 int chkPlayerWin();
00130
00147 void clearGrid();
00148
00165 void updateScoreBtn(gpointer data);
00166
00192 void on_btnGrid_clicked(GtkWidget *widget, gpointer data);
00193
00204 void on_btnScore_clicked(GtkWidget *widget, gpointer data);
00214 void showWin();
00215
00216 #endif // MAIN_H // End of include guard
```

6.10 header/minimax.h File Reference

Header file for the Tic-Tac-Toe Minimax algorithm.

```
#include <macros.h>
#include <elapsedTime.h>
```

Data Structures

struct BoardState

Stores the current state of the Tic-Tac-Toe board along with the best move.

Macros

- #define FILE_BESTMOV "resources/bestmove.txt"
- #define MAX_BOARDS 10000

Functions

• int max (int a, int b)

Returns the maximum of two integers.

• int min (int a, int b)

Returns the minimum of two integers.

struct Position findBestMove (int board[3][3])

Finds the best move for the bot in the Tic-Tac-Toe game.

int minimax (int board[3][3], int depth, bool isMax)

Implements the Minimax algorithm to evaluate the best move for the bot.

• int evaluate (int b[3][3])

Evaluates the current board state to determine if there is a winner.

bool isMovesLeft (int board[3][3])

Checks if there are any moves left on the board.

bool checkAndUpdateBestMove (int board[3][3], struct Position *bestMove, struct BoardState boardStates[], int count)

Checks if the current board configuration exists in the lookup table and updates the best move.

• void writeBestMoveToFile (int board[3][3], struct Position bestMove)

Appends the current board state and the best move to a file.

• int loadBoardStates (struct BoardState boardStates[])

Loads board states and their best moves from a file.

• void printFileContents ()

Prints the contents of the best move file.

6.10.1 Detailed Description

Header file for the Tic-Tac-Toe Minimax algorithm.

Author

jacktan-jk

Version

1.0

Date

2024-11-12

Copyright

Copyright (c) 2024

This file contains function declarations, structure definitions, and constants for implementing the Minimax algorithm in a Tic-Tac-Toe game. It includes functions for evaluating board states, calculating the best move for the bot, and checking if any moves are left on the board.

Definition in file minimax.h.

6.10.2 Macro Definition Documentation

FILE BESTMOV

```
#define FILE_BESTMOV "resources/bestmove.txt"
Path to the file storing best moves
Definition at line 35 of file minimax.h.
```

MAX_BOARDS

```
#define MAX_BOARDS 10000

Maximum number of boards to store in memory

Definition at line 36 of file minimax.h.
```

6.10.3 Function Documentation

checkAndUpdateBestMove()

```
struct BoardState boardStates[],
int count)
```

Checks if the current board configuration exists in the lookup table and updates the best move.

This function compares the current board with previously saved board states in the boardStates array. If a matching board configuration is found, it updates the provided bestMove structure with the best move associated with that board state. The function returns true if a match is found and the move is updated, and false if no match is found in the lookup table.

Parameters

board	The current Tic Tac Toe board to check against the saved states.
bestMove	A pointer to the Position structure where the best move will be stored if a match is found.
boardStates	An array of BoardState structures containing previously saved board configurations and their
	best moves.
count	The number of saved board states in the boardStates array.

Returns

true if a matching board configuration is found and the best move is updated, false otherwise.

See also

BoardState, Position

Definition at line 419 of file minimax.c.

evaluate()

```
int evaluate ( int b[3][3])
```

Evaluates the current board state to determine if there is a winner.

This function checks the Tic-Tac-Toe board for winning conditions, i.e., it checks rows, columns, and diagonals for three consecutive marks (either BOT or PLAYER1). It returns a score based on the result:

- +10 if the BOT wins.
- -10 if PLAYER1 wins.
- 0 if there is no winner yet (no winner in rows, columns, or diagonals).

Parameters

b A 3x3 array representing the Tic-Tac-Toe board.

Returns

The evaluation score:

- +10 for a BOT win,
- -10 for a PLAYER1 win,
- 0 if there is no winner.

See also

BOT, PLAYER1

Definition at line 220 of file minimax.c.

findBestMove()

Finds the best move for the bot in the Tic-Tac-Toe game.

This function first checks if the best move is already stored in memory by looking through previous board states. If the move is found, it is returned. If no best move is found in memory, it traverses all the empty cells on the board, evaluates the potential moves using the minimax algorithm, and returns the optimal move.

Parameters

```
board A 3x3 array representing the current Tic-Tac-Toe board.
```

Returns

The best move for the bot as a struct Position containing the row and column.

See also

minimax, loadBoardStates, checkAndUpdateBestMove, writeBestMoveToFile

Definition at line 71 of file minimax.c.

isMovesLeft()

```
bool isMovesLeft (
          int board[3][3])
```

Checks if there are any moves left on the board.

This function determines if there are any empty cells left on a 3x3 board. It uses one of the following implementations based on compilation options:

- A standard C implementation when DISABLE_ASM is defined.
- Optimized assembly instructions for either AArch64 or x86 platforms.

The choice of implementation depends on the platform and preprocessor directives.

Parameters

board

A 3x3 array representing the board state. Each cell should contain:

- EMPTY (typically 0) if the cell is empty.
- · Any non-zero value if the cell is occupied.

Returns

true if there are empty cells; otherwise, false.

Note

If DISABLE_ASM is defined, the function uses pure C logic.

For AArch64 platforms, the function uses assembly instructions for efficient scanning.

For x86 platforms, the function uses assembly instructions for efficient scanning.

Example usage:

```
int board[3][3] = {
     {1, 2, 0},
     {0, 1, 2},
     {2, 1, 0}
};
bool movesLeft = isMovesLeft(board);
// movesLeft will be true as there are empty cells (0s).
```

Definition at line 270 of file minimax.c.

loadBoardStates()

Loads board states and their best moves from a file.

This function attempts to open a file containing saved board states and the corresponding best move for each state. If the file does not exist, a new file is created. It reads the board configurations and the best move for each board, storing them in the provided boardStates array.

Each line in the file represents one board state. The board is stored as a 3x3 grid, where 'x' denotes the BOT's move, 'o' denotes PLAYER1's move, and empty spaces are represented as ' ' (empty). The best move for each board is also saved in the file.

Parameters

	boardStates	An array of BoardState structures to store the loaded board states.
--	-------------	---------------------------------------------------------------------

Returns

The number of boards loaded from the file. If the file does not exist, it returns 0 and creates a new file.

See also

```
BoardState, FILE_BESTMOV
```

Definition at line 366 of file minimax.c.

max()

Returns the maximum of two integers.

This function computes the maximum of two integers using either:

- A standard C implementation when DISABLE_ASM is defined.
- Optimized assembly instructions for either AArch64 or x86 platforms.

The choice of implementation depends on the platform and preprocessor directives.

Parameters

а	The first integer to compare.
b	The second integer to compare.

Returns

The greater of the two integers.

Note

If DISABLE_ASM is defined, the function uses pure C logic.

For AArch64 platforms, the function uses assembly instructions with conditional selection.

For x86 platforms, the function uses assembly instructions with register manipulation and branching.

Example usage:

```
int result = max(10, 20);

// result now holds the value 20.

Definition at line 5 of file minimax.c.
```

min()

Returns the minimum of two integers.

This function computes the minimum of two integers using either:

- A standard C implementation when DISABLE_ASM is defined.
- Optimized assembly instructions for either AArch64 or x86 platforms.

The choice of implementation depends on the platform and preprocessor directives.

Parameters

а	The first integer to compare.
b	The second integer to compare.

Returns

The smaller of the two integers.

Note

If DISABLE_ASM is defined, the function uses pure C logic.

For AArch64 platforms, the function uses assembly instructions with conditional selection.

For x86 platforms, the function uses assembly instructions with register manipulation and branching.

Example usage:

```
int result = min(10, 20);
// result now holds the value 10.
Definition at line 38 of file minimax.c.
```

minimax()

```
int minimax (
    int board[3][3],
    int depth,
    bool isMax)
```

Implements the Minimax algorithm to evaluate the best move for the bot.

The function recursively evaluates all possible moves using the Minimax algorithm. It returns the best score for the current player (maximizer or minimizer) based on the game state. The algorithm chooses the optimal move for the bot and evaluates the game state at each depth. The depth is capped if Minimax Godmode is not enabled. If there are no moves left or the game is over, it returns the evaluation score.

Parameters

board	A 3x3 array representing the current Tic-Tac-Toe board.
depth	The current depth in the game tree.
isMax	Boolean flag indicating whether it is the maximizer's turn (bot) or the minimizer's turn (player).

Returns

The best score for the current move based on the evaluation function.

See also

```
evaluate, isMovesLeft, max, min
```

Definition at line 136 of file minimax.c.

printFileContents()

```
void printFileContents ()
```

Prints the contents of the best move file.

This function reads and prints the contents of the file that stores the best moves for various board states.

writeBestMoveToFile()

```
void writeBestMoveToFile (
         int board[3][3],
         struct Position bestMove)
```

Appends the current board state and the best move to a file.

This function writes the current Tic Tac Toe board state to a file, encoding the board as a sequence of characters where 'o' represents Player 1, 'x' represents the Bot, and 'b' represents an empty cell. After writing the board state, it appends the best move (row and column) for the current board to the same file.

Parameters

board	The current Tic Tac Toe board to write to the file.]
bestMove	The best move to be made, represented by its row and column indices.	Ī

See also

Position, BoardState

Definition at line 436 of file minimax.c.

6.11 minimax.h

Go to the documentation of this file.

```
00001
00016 #ifndef MINIMAX_H // Start of include guard
00017 #define MINIMAX_H
00018
00019 #include <macros.h>
00020 #include <elapsedTime.h>
00021
00029 struct BoardState
00030 {
00031
          int board[3][3];
00032
         struct Position bestMove;
00033 };
00034
00035 #define FILE_BESTMOV "resources/bestmove.txt"
00036 #define MAX_BOARDS 10000
00061 int max(int a, int b);
00062
00086 int min(int a, int b);
00087
00103 struct Position findBestMove(int board[3][3]);
00104
00123 int minimax(int board[3][3], int depth, bool isMax);
00124
00144 int evaluate(int b[3][3]);
00145
00177 bool isMovesLeft(int board[3][3]);
00178
00196 bool checkAndUpdateBestMove(int board[3][3], struct Position *bestMove, struct BoardState
     boardStates[], int count);
00197
00210 void writeBestMoveToFile(int board[3][3], struct Position bestMove);
00211
00232 int loadBoardStates(struct BoardState boardStates[]);
00233
00240 void printFileContents();
00241
00242 #endif // MINIMAX_H // End of include guard
```

6.12 header/ml-naive-bayes.h File Reference

Header file for Naive Bayes classifier functions for Tic-Tac-Toe outcome prediction.

```
#include <macros.h>
#include <importData.h>
```

Macros

- #define TRAINING_DATA_SIZE 0.8 * DATA_SIZE
- #define TESTING DATA SIZE 0.2 * DATA SIZE
- #define CLASSES 2

Functions

• int assignMoveIndex (char move)

Assigns an index to each move ("x", "o", or "b").

void calculateProbabilities (int dataset_size)

Calculates the probabilities for each class and conditional probabilities with Laplace smoothing.

void resetTrainingData ()

Resets the training data and associated statistics for a fresh training cycle.

• int initData ()

Initializes the training data and model statistics.

int predictOutcome (struct Dataset board)

Predicts the outcome of a given Tic Tac Toe board based on previously calculated probabilities.

void calcTrainErrors ()

Calculates the training errors and the probability of error.

void calcConfusionMatrix ()

Calculates the confusion matrix and error probability for the testing dataset.

struct Position getBestPosition (int grid[3][3], char player)

Determines the best position for the bot to make a move based on the highest probability.

int getTruthValue (char *str1)

Returns an integer value representing a truth value based on input.

void assignCMValue (int actual, int predicted)

Updates the confusion matrix based on actual and predicted outcomes.

void debugDataset (struct Dataset *data, int len)

Debug function to display dataset contents.

6.12.1 Detailed Description

Header file for Naive Bayes classifier functions for Tic-Tac-Toe outcome prediction.

Author

jacktan-jk

Version

1.0

Date

2024-11-13

Copyright

Copyright (c) 2024

This file contains function declarations, structure definitions, and global variables for implementing the Naive Bayes classification model for predicting outcomes in a Tic-Tac-Toe game. It includes functions for initializing training data, calculating probabilities, predicting outcomes based on the trained model, calculating error rates, and updating the confusion matrix. The model is trained using a dataset of game board states and outcomes, and is used to predict the outcome of new game states.

Definition in file ml-naive-bayes.h.

6.12.2 Macro Definition Documentation

CLASSES

```
#define CLASSES 2
```

Number of possible outcome classes (positive/negative)

Definition at line 26 of file ml-naive-bayes.h.

TESTING_DATA_SIZE

```
#define TESTING_DATA_SIZE 0.2 * DATA_SIZE Size of the testing dataset (20%)
Definition at line 25 of file ml-naive-bayes.h.
```

TRAINING DATA SIZE

```
#define TRAINING_DATA_SIZE 0.8 * DATA_SIZE Size of the training dataset (80%)
Definition at line 24 of file ml-naive-bayes.h.
```

6.12.3 Function Documentation

assignCMValue()

```
void assignCMValue (
    int actual,
    int predicted)
```

Updates the confusion matrix based on actual and predicted outcomes.

This function updates the confusion matrix counters for true positives, false negatives, false positives, and true negatives. It checks the actual and predicted outcomes and increments the appropriate counter in the confusion matrix.

If either the actual or predicted value is ERROR, an error is logged.

Parameters

actual	The actual outcome value (1 for positive, 0 for negative).
predicted	The predicted outcome value (1 for positive, 0 for negative).

See also

cM, ERROR

Definition at line 340 of file ml-naive-bayes.c.

assignMoveIndex()

Assigns an index to each move ("x", "o", or "b").

This function maps the board move characters to their corresponding integer values:

- · 'x' is mapped to the BOT.
- 'o' is mapped to PLAYER1.
- 'b' is mapped to EMPTY. If the character does not match any of the valid moves, -1 is returned.

Parameters

move	The character representing the move ('x', 'o', or 'b').
------	---------------------------------------------------------

Returns

int The integer corresponding to the move:

- · BOT for 'x',
- · PLAYER1 for 'o',
- · EMPTY for 'b',
- · ERROR for invalid input.

See also

```
BOT, PLAYER1, EMPTY, ERROR
```

Definition at line 21 of file ml-naive-bayes.c.

calcConfusionMatrix()

```
void calcConfusionMatrix ()
```

Calculates the confusion matrix and error probability for the testing dataset.

This function evaluates the model's performance by calculating the confusion matrix based on actual and predicted outcomes. It iterates through the testing data, compares actual outcomes with predicted ones, and updates the confusion matrix values. The number of prediction errors and the probability of error are also computed.

See also

cM, test_PredictedErrors, probabilityErrors, getTruthValue, predictOutcome

Definition at line 373 of file ml-naive-bayes.c.

calcTrainErrors()

```
void calcTrainErrors ()
```

Calculates the training errors and the probability of error.

This function evaluates the model's performance on the training dataset by comparing predicted outcomes with actual ones. It updates the count of prediction errors and computes the probability of error based on the number of errors and the size of the training dataset.

See also

train_PredictedErrors, probabilityErrors, getTruthValue, predictOutcome

Definition at line 424 of file ml-naive-bayes.c.

calculateProbabilities()

Calculates the probabilities for each class and conditional probabilities with Laplace smoothing.

This function calculates:

- The class probabilities for positive and negative outcomes.
- The conditional probabilities for each move ('x', 'o', 'b') at each position on the board, given the class (positive
 or negative) with Laplace smoothing applied.

The Laplace smoothing is used to prevent zero probabilities for moves that may not have been observed in the training data. The resulting probabilities are printed for debugging purposes.

Parameters

The total number of samples in the dataset used for probability calculation.	dataset_size
------------------------------------------------------------------------------	--------------

See also

positive count, negative count, positiveMoveCount, negativeMoveCount

Definition at line 36 of file ml-naive-bayes.c.

debugDataset()

Debug function to display dataset contents.

This function prints the details of the provided dataset, including the grid values and the corresponding outcomes. It is primarily used for debugging purposes and is not currently in use within the code.

Parameters

data	Pointer to the dataset to be printed.
len	The length of the dataset (number of entries).

See also

PRINT_DEBUG

Definition at line 451 of file ml-naive-bayes.c.

getBestPosition()

Determines the best position for the bot to make a move based on the highest probability.

This function evaluates all empty positions on the Tic Tac Toe grid and calculates the probability of the bot winning (either as 'x' or 'o') using the pre-calculated move probabilities from the training data. The bot chooses the position with the highest probability of winning, where the move is either 'x' or 'o' depending on the current player. It returns the best position for the bot to make its move.

Parameters

grid	The current state of the Tic Tac Toe game board.
player	The current player, either 'x' or 'o'.

Returns

A struct Position representing the row and column of the best move for the bot. If no valid move is found, it returns an error indicator.

See also

positive_count, negative_count, positiveMoveCount, negativeMoveCount

Definition at line 156 of file ml-naive-bayes.c.

getTruthValue()

Returns an integer value representing a truth value based on input.

This function converts a string input into a corresponding integer truth value. The input string is compared against "positive" and "negative" to determine the output:

- · Returns 1 for "positive".
- · Returns 0 for "negative".
- Returns −1 for any other input, and logs an error message for invalid cases.

Parameters

str1 A pointer to the input string to be evaluated.

Returns

int The corresponding truth value:

- 1 for "positive".
- 0 for "negative".
- -1 for invalid inputs.

See also

PRINT DEBUG

Definition at line 406 of file ml-naive-bayes.c.

initData()

```
int initData ()
```

Initializes the training data and model statistics.

This function resets the training data, then retrieves the training dataset for model training. It processes the dataset to count occurrences of positive and negative outcomes and updates the move counts for each grid position based on the data. Afterward, it calculates training errors and updates the confusion matrix.

If the initial dataset is empty, it attempts to load the data again.

See also

resetTrainingData, getTrainingData, calcTrainErrors, calcConfusionMatrix

Definition at line 273 of file ml-naive-bayes.c.

predictOutcome()

Predicts the outcome of a given Tic Tac Toe board based on previously calculated probabilities.

This function calculates the probabilities of a positive (Player 1 wins) or negative (Bot wins) outcome for a given board state by multiplying the conditional probabilities of each move in the grid with the class probabilities. The prediction is made based on which outcome (positive or negative) has the higher probability.

If the calculated probabilities are zero, indicating that the outcome cannot be predicted with the available data, the function returns -1.

Parameters

board The current Tic Tac Toe board whose outcome needs to be predicted.

Returns

1 if the predicted outcome is positive (Player 1 wins), 0 if negative (Bot wins), and -1 if the outcome cannot be predicted.

See also

positiveClassProbability, negativeClassProbability, positiveMoveCount, negativeMoveCount, assignMoveIndex

Definition at line 88 of file ml-naive-bayes.c.

resetTrainingData()

```
void resetTrainingData ()
```

Resets the training data and associated statistics for a fresh training cycle.

This function resets all relevant variables used in the machine learning model's training process. It clears the outcome counts, resets the move count arrays for each grid position, and reinitializes the confusion matrix. Additionally, it clears the prediction error counters, ensuring that the model starts with a clean state.

See also

positive_count, negative_count, positiveMoveCount, negativeMoveCount, cM, test_PredictedErrors, train PredictedErrors

Definition at line 247 of file ml-naive-bayes.c.

6.13 ml-naive-bayes.h

Go to the documentation of this file.

```
00018 #ifndef ML_NAIVE_BAYES_H
00019 #define ML_NAIVE_BAYES_H
00020
00021 #include <macros.h>
00022 #include <importData.h>
00023
00024 #define TRAINING_DATA_SIZE 0.8 * DATA_SIZE
00025 #define TESTING_DATA_SIZE 0.2 * DATA_SIZE
00026 #define CLASSES 2
00084 int assignMoveIndex(char move);
00085
00101 void calculateProbabilities(int dataset_size);
00102
00112 void resetTrainingData();
00113
00125 int initData();
00126
00143 int predictOutcome(struct Dataset board);
00144
00153 void calcTrainErrors();
00154
00164 void calcConfusionMatrix();
00165
00181 struct Position getBestPosition(int grid[3][3], char player);
00182
00201 int getTruthValue(char *str1);
00202
00216 void assignCMValue(int actual, int predicted);
00217
00229 void debugDataset(struct Dataset *data, int len);
00231 #endif // ML_NAIVE_BAYES_H
```

6.14 mainpage.md File Reference

6.15 src/elapsedTime.c File Reference

```
#include <elapsedTime.h>
```

Functions

void startElapseTime ()

Starts the elapsed time tracking.

void stopElapseTime (char *str)

Stops the elapsed time tracking and outputs the result.

Variables

- struct timeval gTime
- double gStartTime
- double gEndTime

6.16 elapsedTime.c 35

6.15.1 Function Documentation

startElapseTime()

```
void startElapseTime ()
```

Starts the elapsed time tracking.

Captures the current time and stores it in gStartTime to mark the beginning of an elapsed time measurement. Only operates if DISABLE_ELAPSED is not defined, allowing conditional compilation for performance tracking. Definition at line 9 of file elapsedTime.c.

stopElapseTime()

Stops the elapsed time tracking and outputs the result.

Calculates the time elapsed since startElapseTime and outputs it in seconds using the provided label.

Parameters

str Label describing the operation or section being timed.

Only operates if <code>DISABLE_ELAPSED</code> is not defined, and outputs timing information through <code>PRINT_DEBUG</code> for profiling and debugging.

Definition at line 17 of file elapsedTime.c.

6.15.2 Variable Documentation

gEndTime

double gEndTime

Holds the end time in seconds for the timed section.

Definition at line 6 of file elapsedTime.c.

gStartTime

```
double gStartTime
```

Holds the start time in seconds for the timed section.

Definition at line 5 of file elapsedTime.c.

gTime

```
struct timeval gTime
```

Stores the current time values for elapsed time calculation.

Definition at line 4 of file elapsedTime.c.

6.16 elapsedTime.c

Go to the documentation of this file.

```
00001 #include <elapsedTime.h>
00003 #if !(DISABLE_ELAPSED)
00004 struct timeval gTime;
00005 double gStartTime;
00006 double gEndTime;
00007 #endif
80000
00009 void startElapseTime()
00010 {
00011 #if !(DISABLE_ELAPSED)
         gettimeofday(&gTime, NULL);
00012
          gStartTime = gTime.tv_sec + 1.0e-6 * gTime.tv_usec;
00013
00014 #endif
00015 }
00016
00017 void stopElapseTime(char *str)
00018 {
```

6.17 src/importData.c File Reference

```
#include <importData.h>
```

Functions

int readDataset (const char *filename, bool split)

Reads a dataset from a file and optionally randomizes entries for training and testing.

• int splitFile ()

Splits the dataset into training and testing files with an 80-20 ratio.

void getRandomNo (int random[DATA SIZE])

Generates an array of unique random integers within the range of the dataset size.

int getTrainingData (struct Dataset **d)

Retrieves the training data from a file and returns its length.

int getTestingData (struct Dataset **d)

Retrieves the testing data from a file and returns its length.

Variables

• int len_train = 0

Global variable to store the number of training dataset entries.

• int len_test = 0

Global variable to store the number of testing dataset entries.

int randomNo [DATA_SIZE]

Global array to store unique random indices for dataset splitting.

struct Dataset data [DATA_SIZE]

Global array to store the dataset.

const char * trainingFile = RES_PATH "" TRAIN_PATH "" DATA_PATH

Global variable to store the path for the training dataset file.

• const char * testingFile = RES PATH "" TEST PATH "" DATA PATH

Global variable to store the path for the testing dataset file.

6.17.1 Function Documentation

getRandomNo()

Generates an array of unique random integers within the range of the dataset size.

This function populates an array with unique random integers between 0 and DATA_SIZE - 1. It ensures that each integer appears only once by checking a check array to track used indices. This can be used for randomizing the order of data for splitting purposes.

Parameters

random	Array to store the generated unique random integers.
--------	------------------------------------------------------

See also

DATA_SIZE

Definition at line 143 of file importData.c.

getTestingData()

Retrieves the testing data from a file and returns its length.

This function zeroes out the data array for the length of the testing set, reads the dataset from the specified testingFile, and assigns the data pointer to the provided dataset pointer. Returns the length of the testing data loaded.

Parameters

d Pointer to a dataset pointer that will reference the loaded testing data array.

Returns

The number of testing entries loaded (i.e., len_test).

See also

readDataset, testingFile

Definition at line 175 of file importData.c.

getTrainingData()

Retrieves the training data from a file and returns its length.

This function initializes the data array to zero for the length of the training set, reads the dataset from the specified trainingFile, and assigns the data pointer to the provided dataset pointer. Returns the length of the training data loaded.

Parameters

d Pointer to a dataset pointer that will reference the loaded training data array.

Returns

The number of training entries loaded (i.e., len_train).

See also

readDataset, trainingFile

Definition at line 167 of file importData.c.

readDataset()

Reads a dataset from a file and optionally randomizes entries for training and testing.

Opens a file to read each line as a Tic Tac Toe board state, populating a grid structure where 'x', 'o', and 'b' represent the Bot, Player 1, and empty cells, respectively. Each board state is followed by an outcome that is stored within the dataset. If split is true, entries are randomized using an array of unique indices for shuffling.

Parameters

filename	The name of the dataset file to read.]
split	Boolean indicating whether to randomize entries for dataset splitting.	

Returns

int SUCCESS (0) if reading is successful, BAD_PARAM (-5) if the file cannot be opened, or the return value of splitFile() if split is enabled.

See also

getRandomNo, splitFile

Definition at line 44 of file importData.c.

splitFile()

```
int splitFile ()
```

Splits the dataset into training and testing files with an 80-20 ratio.

This function separates the dataset into two parts: 80% for training and 20% for testing. The training portion is written to trainingFile, and the testing portion is written to testingFile. Each entry consists of a 3x3 grid representing the Tic Tac Toe board and the outcome of that board.

Returns

int SUCCESS (0) if both files are written successfully, BAD_PARAM (-5) if either file cannot be opened.

See also

data, trainingFile, testingFile

Definition at line 90 of file importData.c.

6.17.2 Variable Documentation

data

```
struct Dataset data[DATA_SIZE]
```

Global array to store the dataset.

This array holds the Tic-Tac-Toe board states and their corresponding outcomes.

Definition at line 38 of file importData.c.

len_test

```
int len_test = 0
```

Global variable to store the number of testing dataset entries.

This variable tracks the size of the testing dataset after splitting.

Definition at line 36 of file importData.c.

len_train

```
int len_train = 0
```

Global variable to store the number of training dataset entries.

This variable tracks the size of the training dataset after splitting.

Definition at line 35 of file importData.c.

randomNo

```
int randomNo[DATA_SIZE]
```

Global array to store unique random indices for dataset splitting.

This array stores randomized indices used to split the dataset into training and testing subsets.

Definition at line 37 of file importData.c.

testingFile

```
const char* testingFile = RES_PATH "" TEST_PATH "" DATA_PATH
```

Global variable to store the path for the testing dataset file.

This variable holds the full path to the testing dataset file for reading and writing.

Definition at line 42 of file importData.c.

6.18 importData.c 39

trainingFile

```
const char* trainingFile = RES_PATH "" TRAIN_PATH "" DATA_PATH
```

Global variable to store the path for the training dataset file.

This variable holds the full path to the training dataset file for reading and writing. Definition at line 41 of file importData.c.

6.18 importData.c

Go to the documentation of this file.

```
00001 #include <importData.h
00002
00035 int len_train = 0;
00036 int len_test = 0;
00037 int randomNo[DATA_SIZE];
00038 struct Dataset data[DATA_SIZE];
00039
00040 // to write to directory before
00041 const char *trainingFile = RES_PATH "" TRAIN_PATH "" DATA_PATH;
00042 const char *testingFile = RES_PATH "" TEST_PATH "" DATA_PATH;
00043
00044 int readDataset(const char *filename, bool split)
00045 {
          FILE *file = fopen(filename, "r");
00046
00047
          if (!file)
00048
              PRINT_DEBUG("[ERROR] Error opening file.\n");
              return BAD_PARAM;
00050
00051
          }
00052
00053
          if (split)
00054
00055
              // get an array of random int where each position is different
00056
              getRandomNo(randomNo);
00057
00058
00059
          char line[100];
          for (int i = 0; i < DATA_SIZE && fgets(line, sizeof(line), file); i++)</pre>
00060
00061
00062
               // Get first token with delimiter being ","
00063
              char *token = strtok(line, ",");
00064
              for (int row = 0; row < 3; row++)</pre>
00065
00066
                   for (int col = 0; col < 3; col++)</pre>
00067
00068
                       if (token != NULL)
00069
                           data[split ? randomNo[i] : i].grid[row][col] = token[0];
token = strtok(NULL, ",");
00070
00071
00072
00073
                   }
              }
00075
00076
              if (token != NULL)
00077
              {
00078
                  strncpy(data[split ? randomNo[i] : i].outcome, token, sizeof(data[split ? randomNo[i] :
      i].outcome) - 1);
00079
08000
00081
          fclose(file);
00082
          if (split)
00083
00084
          {
00085
              return splitFile();
00086
00087
          return SUCCESS;
00088 }
00089
00090 int splitFile()
00091 {
          // get 80% and 20% respectively
00093
          int eighty = len_train = 0.8 * DATA_SIZE;
          len_test = 0.2 * DATA_SIZE;
00094
00095
00096
           // write into training dataset
          FILE *trainFile;
00097
00098
          trainFile = fopen(trainingFile, "w");
00099
           if (!trainFile)
00100
00101
              PRINT_DEBUG("[ERROR] Error opening file.\n");
00102
              return BAD_PARAM;
00103
          }
00104
```

```
00105
          for (int i = 0; eighty > i; i++)
00106
00107
               for (int row = 0; 3 > row; row++)
00108
00109
                   for (int col = 0; 3 > \text{col}; col++)
00110
00111
                       fprintf(trainFile, "%c,", data[i].grid[row][col]);
00112
00113
               fprintf(trainFile, "%s\n", data[i].outcome);
00114
          }
00115
00116
00117
          fclose(trainFile);
00118
00119
          // write into testing dataset
          FILE *testFile;
testFile = fopen(testingFile, "w");
00120
00121
          if (!testFile)
00122
00123
00124
              PRINT_DEBUG("[ERROR] Error opening file.\n");
00125
              return BAD_PARAM;
00126
          }
00127
          for (int i = eighty; DATA_SIZE > i; i++)
00128
00129
00130
               for (int row = 0; 3 > row; row++)
00131
00132
                   for (int col = 0; 3 > col; col++)
00133
                       fprintf(testFile, "%c,", data[i].grid[row][col]);
00134
00135
                   }
00136
00137
               fprintf(testFile, "%s\n", data[i].outcome);
00138
00139
          fclose(testFile);
00140
          return SUCCESS;
00141 }
00142
00143 void getRandomNo(int random[DATA_SIZE])
00144 {
00145
          int count = 0:
00146
          srand(time(NULL));
00147
00148
          // initialize all to 0 for proper check
00149
          int check[DATA_SIZE];
00150
          for (int i = 0; DATA_SIZE > i; i++)
00151
               check[i] = 0;
00152
          }
00153
00154
00155
          while (DATA_SIZE > count)
00156
          {
00157
               int randNo = rand() % DATA_SIZE;
00158
               if (check[randNo] == 0)
               {
00159
00160
                   check[randNo] = 1;
random[count] = randNo;
00161
00162
                   count++;
00163
00164
          }
00165 }
00166
00167 int getTrainingData(struct Dataset **d)
00168 {
00169
          memset(data, 0, len_train * sizeof(struct Dataset));
00170
          readDataset(trainingFile, false);
00171
          *d = data;
00172
          return len train:
00173 }
00174
00175 int getTestingData(struct Dataset **d)
00176 {
          memset(data, 0, len_test * sizeof(struct Dataset));
readDataset(testingFile, false);
00177
00178
00179
          *d = data;
00180
          return len_test;
00181 }
```

6.19 src/main.c File Reference

```
#include <main.h>
```

Functions

· void clearGrid ()

Clears the game board and resets the player's turn.

· void updateScoreBtn (gpointer data)

Updates the score display on the button.

void on_btnGrid_clicked (GtkWidget *widget, gpointer data)

Callback function for handling button clicks on the game grid.

void on_btnScore_clicked (GtkWidget *widget, gpointer data)

Handles button click for score.

• void showWin ()

Clears the winning positions and resets the grid.

• int doBOTmove ()

Executes the bot's move based on the current game mode.

• int chkPlayerWin ()

Checks the current game board for a win or tie.

• int main (int argc, char *argv[])

Initializes and runs the Tic-Tac-Toe GTK application.

Variables

• int iPlayer1_score = 0

Global variable to track Player 1's score.

• int iPlayer2 score = 0

Global variable to track Player 2's or Bot's (Minimax/ML) score.

• int iTie_score = 0

Global variable to track the number of ties/draws.

int iGameState = PLAY

Global variable to track the current game state.

• int iBoard [3][3]

Global 2D array representing the Tic-Tac-Toe game board.

int iWinPos [3][3]

Global 2D array to track winning positions on the board.

• bool isPlayer1Turn = true

Global flag indicating if it's Player 1's turn.

• bool isMLAvail = true

Global flag indicating if Machine Learning mode is available. This is set to false if the ML data file is missing, disabling the ML game mode.

struct PlayerMode playerMode = {"2P", MODE_2P}

Global structure to track the current game mode.

GtkWidget * btnGrid [3][3]

Global 2D array of buttons corresponding to the game grid.

6.19.1 Function Documentation

chkPlayerWin()

```
int chkPlayerWin ()
```

Checks the current game board for a win or tie.

This function checks all possible win conditions:

- · Diagonals
- Rows

· Columns

If there is a winning line, it marks the winning positions and returns WIN. If there are no winning conditions and the board is full, it returns TIE. If there are unclicked positions left, it returns PLAY.

Returns

WIN if there is a winner, TIE if the game is a tie, PLAY if the game is still ongoing.

See also

```
iBoard, iWinPos
```

Definition at line 234 of file main.c.

clearGrid()

```
void clearGrid ()
```

Clears the game board and resets the player's turn.

This function is used to reset the game board for a new round. It clears the labels on the buttons in the grid and resets the internal board state (iBoard) to 0. It also sets the player turn back to player 1.

- Sets all button labels in the btnGrid to an empty string.
- Resets all values in the iBoard array to 0, indicating no moves.
- Resets isPlayer1Turn to true, indicating it's Player 1's turn.

See also

iBoard

btnGrid

isPlayer1Turn

Definition at line 29 of file main.c.

doBOTmove()

```
int doBOTmove ()
```

Executes the bot's move based on the current game mode.

In MM mode:

- · Performs a minimax move.
- 20% chance of the minimax randomly selects a position.

In ML mode, the bot uses machine learning to determine the best position.

The function also measures and logs the time taken for the minimax move.

Returns

SUCCESS if the bot's move was made successfully.

See also

```
playerMode, isMLAvail, iBoard, findBestMove, getBestPosition, btnGrid
```

Definition at line 181 of file main.c.

main()

Initializes and runs the Tic-Tac-Toe GTK application.

This function initializes GTK, creates the main window, and sets up the game grid, score display, and buttons. It also handles the setup for the game mode and ML availability. The game board and score are displayed, and event listeners are attached to buttons.

Parameters

argc	The number of arguments passed to the program.
argv	The list of arguments passed to the program.

Returns

SUCCESS if the program runs successfully.

See also

```
initData, on_btnScore_clicked, on_btnGrid_clicked, btnGrid
```

Definition at line 302 of file main.c.

on_btnGrid_clicked()

Callback function for handling button clicks on the game grid.

This function handles the logic for a player's move when a button in the game grid is clicked. It updates the game state, checks for a winner or tie, and updates the score display. It also handles player turns, Bot moves (if applicable), and resets the game board when the game state changes.

Parameters

widget	The GtkWidget that was clicked (the button in the grid).
data	Additional data passed to the callback (usually the score display data).

- If the game state is not PLAY, the game will be reset, and the score updated.
- If the clicked button already has a label, the function returns early (no action is taken).
- If the clicked button is empty, the move is recorded in the iBoard array (Player 1 or MM or ML).
- After each move, the game checks for a win or tie condition using chkPlayerWin().
- If Player 1 or Player 2 wins, the score is updated, and the win condition is shown.
- If the game ends in a tie, the tie score is updated.
- If the game is in **2P** mode, turns alternate between Player 1 and Player 2.
- In MM mode, the Minimax will automatically make a move after Player 1's turn.
- In ML mode, the dataset is re-read and initialized after the game ends.

See also

```
iBoard, isPlayer1Turn, iPlayer1_score, iPlayer2_score, iTie_score playerMode, updateScoreBtn, chkPlayerWin, doBOTmove, showWin PLAY, TIE, WIN
```

Definition at line 60 of file main.c.

on_btnScore_clicked()

Handles button click for score.

Toggles the player mode and updates the displayed score.

Parameters

widget	The widget that triggered the event.
data	Additional data passed to the callback.

See also

playerMode, isMLAvail, isPlayer1Turn, updateScoreBtn, clearGrid

Definition at line 131 of file main.c.

showWin()

```
void showWin ()
```

Clears the winning positions and resets the grid.

Iterates over the win positions and clears any displayed labels, resetting the grid to its initial state.

See also

iWinPos, btnGrid

Definition at line 159 of file main.c.

updateScoreBtn()

Updates the score display on the button.

This function updates the label on a score button to display the current scores for Player 1, Player 2, and Ties. It changes the text formatting depending on which player's turn it is, highlighting the active player.

Parameters

- The function checks if it's Player 1's turn and updates the score display with a bold label for Player 1, or Player 2's turn with Player 2's score in bold.
- The button text is updated using gtk_button_set_label(), and the label markup is updated using gtk_label_set_markup().
- The score includes Player 1's score, Player 2's score, the tie count, and the current game mode (player↔ Mode.txt).

See also

```
iPlayer1_score, iTie_score, iPlayer2_score, playerMode
```

Definition at line 42 of file main.c.

6.19.2 Variable Documentation

btnGrid

```
GtkWidget* btnGrid[3][3]
```

Global 2D array of buttons corresponding to the game grid.

Definition at line 19 of file main.c.

iBoard

```
int iBoard[3][3]
```

Global 2D array representing the Tic-Tac-Toe game board.

Definition at line 11 of file main.c.

iGameState

```
int iGameState = PLAY
```

Global variable to track the current game state.

Game states:

- · PLAY: The game is ongoing.
- TIE: The game ended in a tie.
- · WIN: A player has won the game.

Definition at line 10 of file main.c.

iPlayer1_score

```
int iPlayer1_score = 0
```

Global variable to track Player 1's score.

Definition at line 7 of file main.c.

iPlayer2 score

```
int iPlayer2_score = 0
```

Global variable to track Player 2's or Bot's (Minimax/ML) score.

Definition at line 8 of file main.c.

isMLAvail

```
bool isMLAvail = true
```

Global flag indicating if Machine Learning mode is available. This is set to false if the ML data file is missing, disabling the ML game mode.

Definition at line 15 of file main.c.

isPlayer1Turn

```
bool isPlayer1Turn = true
```

Global flag indicating if it's Player 1's turn.

Definition at line 14 of file main.c.

iTie_score

```
int iTie_score = 0
```

Global variable to track the number of ties/draws.

Definition at line 9 of file main.c.

iWinPos

```
int iWinPos[3][3]
```

Global 2D array to track winning positions on the board.

Definition at line 12 of file main.c.

playerMode

```
struct PlayerMode playerMode = {"2P", MODE_2P}
```

Global structure to track the current game mode.

Fields:

- txt: Text representation of the current mode (e.g., "2P", "MM", "ML").
- mode: Integer value representing the current game mode.

Player modes:

- · MODE_2P: Player vs Player mode.
- · MODE MM: Minimax Bot mode.
- · MODE_ML: Machine Learning Bot mode.

Definition at line 17 of file main.c.

6.20 main.c

Go to the documentation of this file.

```
00001 #include <main.h>
00002
00003 /*====
00004 GLOBAL DECLARATION
00005 ========
00006
00007 int iPlayer1_score = 0;
00008 int iPlayer2_score = 0;
00009 int iTie_score = 0;
00010 int iGameState = PLAY;
00011 int iBoard[3][3];
00012 int iWinPos[3][3];
00013
00014 bool isPlayer1Turn = true;
00015 bool isMLAvail = true;
00016
00017 struct PlayerMode playerMode = {"2P", MODE_2P};
00018
00019 GtkWidget *btnGrid[3][3];
00021 /*===========
00022 END OF GLOBAL DECLARATION
00023 ========
00024
00025 /*==
00026 GUI FUNCTIONS
00028
00029 void clearGrid()
00030 {
00031
         isPlayer1Turn = true;
00032
         for (int i = 0; i < 3; i++)
00033
00034
            for (int j = 0; j < 3; j++)
00035
            {
00036
                gtk_button_set_label(GTK_BUTTON(btnGrid[i][j]), ""); // Clear the button labels
00037
                iBoard[i][j] = 0;
00038
            }
00039
         }
00040 }
00041
00042 void updateScoreBtn(gpointer data)
00043 {
00044
         // Update the score display
00045
         char score_text[100];
00046
         if (isPlayer1Turn == true)
00047
            00048
     (X): %d | [%s] ", iPlayer1_score, iTie_score, iPlayer2_score, playerMode.txt);
00049
00050
         else
00051
00052
             snprintf(score_text, sizeof(score_text), "Player 1 (0): %d |
                                                                        TIE: %d
     (X): %d</b>
                | [%s] ", iPlayer1_score, iTie_score, iPlayer2_score, playerMode.txt);
00053
00054
         gtk_button_set_label(GTK_BUTTON(data), score_text); // Update the score button label
00055
         gtk_label_set_markup(GTK_LABEL(gtk_bin_get_child(GTK_BIN(data))), score_text);
00056 }
```

6.20 main.c 47

```
00057
00058
00059 // Callback function for button clicks
00060 void on_btnGrid_clicked(GtkWidget *widget, gpointer data)
00061 {
00062
                    const char *current_label = gtk_button_get_label(GTK_BUTTON(widget));
                   BtnPos *btnPos = (BtnPos *)g_object_get_data(G_OBJECT(widget), "button-data");
00063
00064
00065
                    if (iGameState != PLAY)
00066
00067
                            iGameState = PLAY:
00068
                           clearGrid();
00069
                           updateScoreBtn(data);
00070
                           return;
00071
                   }
00072
00073
                    if (strcmp(current_label, "") != 0)
00074
                   {
00075
                            return;
00076
00077
00078
                   iBoard[btnPos->pos[0]][btnPos->pos[1]] = isPlayer1Turn ? PLAYER1 : BOT; // O (1), X(2), BOT is the algorithm of the property of the property
           same as player 2
00079
08000
                    // Update the button text, for example, with an "O"
                   gtk_button_set_label(GTK_BUTTON(widget), isPlayer1Turn ? "O" : "X");
00081
00082
00083
                   int retVal = chkPlayerWin();
00084
00085
                    if (retVal == PLAY)
00086
                    {
00087
                            isPlayer1Turn = !isPlayer1Turn;
00088
                            updateScoreBtn(data);
00089
00090
                            if (playerMode.mode == MODE_2P)
00091
00092
                                   return;
00093
00094
00095
                            doBOTmove();
00096
                            retVal = chkPlayerWin();
00097
                   }
00098
00099
                    if (retVal == WIN)
00100
                   {
00101
                            showWin();
00102
                            PRINT_DEBUG("[DEBUG] GAME RESULT -> %s Win\n", isPlayer1Turn ? "Player 1" : playerMode.mode ==
           MODE_2P ? "Player 2"
00103
           : "BOT");
00104
                            isPlayer1Turn ? iPlayer1_score++ : iPlayer2_score++;
00105
                            iGameState = WIN;
00106
                   }
00107
                   if (retVal == TIE)
00108
00109
                   {
00110
                            PRINT_DEBUG("[DEBUG] GAME RESULT -> TIE\n");
00111
                            iTie_score++;
00112
                            iGameState = TIE;
00113
                   }
00114
00115
                    if (playerMode.mode != MODE 2P)
00116
00117
                            isPlayer1Turn = !isPlayer1Turn;
00118
00119
00120
                   if (isMLAvail && playerMode.mode == MODE_ML)
00121
00122
                            if (retVal == WIN || retVal == TIE)
00123
                            {
00124
                                    readDataset (RES_PATH "" DATA_PATH, true);
00125
                                    initData();
00126
00127
00128
                   updateScoreBtn(data);
00129 }
00130
00131 void on_btnScore_clicked(GtkWidget *widget, gpointer data)
00132 {
00133
                    playerMode.mode = (playerMode.mode > 1 ? MODE 2P : ++playerMode.mode);
                    switch (playerMode.mode)
00134
00135
00136
                    case MODE_MM:
00137
                            strncpy(playerMode.txt, "MM", sizeof(playerMode.txt));
00138
                           break;
00139
00140
                   case MODE_ML:
```

```
00141
             if (isMLAvail)
00142
00143
                 strncpy(playerMode.txt, "ML", sizeof(playerMode.txt));
00144
00145
00146
00147
         default:
00148
             playerMode.mode = MODE_2P;
             strncpy(playerMode.txt, "2P", sizeof(playerMode.txt));
00149
00150
         PRINT_DEBUG("playerMode: %d\n", playerMode.mode);
00151
         isPlayer1Turn = true;
00152
         iPlayer1_score = iPlayer2_score = iTie_score = 0;
00153
00154
00155
         clearGrid();
00156
         updateScoreBtn(data);
00157 }
00158
00159 void showWin()
00160 {
00161
          for (int i = 0; i < 3; i++)
00162
             for (int j = 0; j < 3; j++)
00163
00164
             {
00165
                 if (iWinPos[i][j] != WIN)
00166
00167
                     gtk_button_set_label(GTK_BUTTON(btnGrid[i][j]), "");
00168
00169
             }
00170
         }
00171
         memset(iWinPos, 0, sizeof(iWinPos));
00172 }
00173 /*----
00174 END OF GUI FUNCTIONS
00175 =========
00176
00177 /
00178 LOGIC FUNCTIONS
00179 =====
00180
00181 int doBOTmove()
00182 {
         struct Position botMove:
00183
00184
         if (playerMode.mode == MODE_MM)
00185
00186
             startElapseTime();
00187 #if !(MINIMAX_GODMODE)
             if (rand() % 100 < 80)</pre>
00188
00189 #endif
00190
             {
00191
                 botMove = findBestMove(iBoard);
00192
00193 #if !(MINIMAX_GODMODE)
00194
             else
00195
             {
00196
                 startElapseTime();
00197
                 int randRow = rand() % 3;
00198
                 int randCol = rand() % 3;
00199
                 bool bIsDone = false;
00200
00201
                 while (!bTsDone)
00202
                 {
00203
                     if (iBoard[randRow][randCol] == EMPTY)
00204
00205
                         PRINT_DEBUG("Random Move -> R:%d C:%d\n", randRow, randCol);
                         botMove.row = randRow;
botMove.col = randCol;
00206
00207
00208
                         bIsDone = !bIsDone;
00209
00210
                     else
00211
00212
                         randRow = rand() % 3;
                         randCol = rand() % 3;
00213
00214
00215
00216
                 stopElapseTime("Minimax Random Move");
00217
00218 #endif
             stopElapseTime("Minimax Move");
00219
00220
00221
         {
m else} // ML mode, sets ML as default if for some reason playermode.mode has expected value.
00222
00223
00224
00225
                 botMove = getBestPosition(iBoard, 'x');
00226
00227
         }
```

6.20 main.c 49

```
00228
00229
          iBoard[botMove.row][botMove.col] = BOT;
00230
          gtk_button_set_label(GTK_BUTTON(btnGrid[botMove.row][botMove.col]), "X");
          return SUCCESS:
00231
00232 }
00233
00234 int chkPlayerWin()
00235 {
00236
          // check both dia
          if (iBoard[0][0] == iBoard[1][1] && iBoard[1][1] == iBoard[2][2] && iBoard[0][0] != 0)
00237
00238
          {
00239
              iWinPos[0][0] = iWinPos[1][1] = iWinPos[2][2] = WIN;
00240
              return WIN;
00241
00242
00243
          if (iBoard[0][2] == iBoard[1][1] && iBoard[1][1] == iBoard[2][0] && iBoard[0][2] != 0)
00244
00245
              iWinPos[0][2] = iWinPos[1][1] = iWinPos[2][0] = WIN;
00246
              return WIN;
00247
          }
00248
00249
          // check rows and col
00250
          for (int i = 0; i < 3; i++)
00251
00252
              // Check rows
00253
              if (iBoard[i][0] == iBoard[i][1] && iBoard[i][1] == iBoard[i][2] && iBoard[i][0] != 0)
00254
00255
                  iWinPos[i][0] = iWinPos[i][1] = iWinPos[i][2] = WIN;
00256
                  return WIN;
00257
00258
              // Check columns
00259
              if (iBoard[0][i] == iBoard[1][i] && iBoard[1][i] == iBoard[2][i] && iBoard[0][i] != 0)
00260
00261
                  iWinPos[0][i] = iWinPos[1][i] = iWinPos[2][i] = WIN;
00262
                  return WIN;
00263
              }
00264
          }
00265
00266
          // check for unclicked grid, if none left then tie
00267
          for (int i = 0; i < 3; i++)
00268
00269
              for (int j = 0; j < 3; j++)
00270
00271
                  if (iBoard[i][j] == 0)
00272
                  {
00273
                      return PLAY;
00274
00275
00276
         }
00277
00278
          return TIE;
00279 }
00280
00281 /*====
00282 END OF LOGIC FUNCTIONS
00283 ====
00285 /*====
00286 MAIN
00287 *Init GUI interface and global variable/objects
00288 ===========
00289
00302 int main(int argc, char *argv[])
00303 {
00304
          int retVal = SUCCESS;
00305
         srand(time(NULL));
00306
00307
          retVal = initData();
00308
          if (retVal != SUCCESS) // disable ML
00309
         {
00310
              isMLAvail = false;
00311
00312
         GtkWidget *window;
00313
00314
          GtkWidget *grid;
00315
          GtkWidget *score_button;
00316
00317
          // Initialize GTK
00318
          gtk_init(&argc, &argv);
00319
00320
          // Create a new window
00321
          window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
00322
          gtk_window_set_title(GTK_WINDOW(window), "Tic-Tac-Toe");
00323
          gtk_window_set_default_size(GTK_WINDOW(window), 250, 950);
00324
          \verb|g_signal_connect(window, "destroy", G_CALLBACK(gtk_main_quit), NULL);|\\
00325
00326
          // Create a box to hold the grid and score button with padding
```

```
GtkWidget *box = gtk_box_new(GTK_ORIENTATION_VERTICAL, 0);
00328
          gtk_container_set_border_width(GTK_CONTAINER(box), 50); // Set padding
00329
00330
          // Create a grid to hold the btnGrid
00331
          grid = gtk grid new();
00332
          gtk_box_pack_start(GTK_BOX(box), grid, TRUE, TRUE, 0); // Add grid to the box
00333
00334
           // Set the grid to expand
00335
          gtk_grid_set_row_homogeneous(GTK_GRID(grid), TRUE);
00336
          gtk_grid_set_column_homogeneous(GTK_GRID(grid), TRUE);
00337
00338
          // Set CSS styles
00339
          GtkCssProvider *css_provider = gtk_css_provider_new();
00340
          gtk_css_provider_load_from_data(css_provider,
00341
                                            "window { background-color: black; }\n"
     "button { background-color: black; color: white; border: 3px solid white; font-size: 24px; background-image: none; }\n"
00342
00343
                                            "button:pressed { background-color: darkgray; }\n",
00344
                                            -1, NULL);
00345
          gtk_style_context_add_provider_for_screen(gdk_screen_get_default(),
00346
                                                       GTK_STYLE_PROVIDER(css_provider),
00347
                                                       GTK_STYLE_PROVIDER_PRIORITY_USER);
00348
00349
          // Create a button for the score display
00350
          score_button = gtk_button_new_with_label("");
00351
          gtk_label_set_markup(GTK_LABEL(gtk_bin_get_child(GTK_BIN(score_button))), "<b>Player 1 (0): 0</b>
          TIE: 0 | Player 2 (X): 0 | [2P] ");
g_signal_connect(score_button, "clicked", G_CALLBACK(on_btnScore_clicked), score_button);
00352
00353
          gtk_grid_attach(GTK_GRID(grid), score_button, 0, 3, 3, 1); // Attach score button below the grid
00354
          // Create the 9 btnGrid and add them to the grid
00355
00356
          for (int i = 0; i < 3; i++)
00357
00358
              for (int j = 0; j < 3; j++)
00359
                  btnGrid[i][j] = gtk_button_new_with_label("");
00360
00361
00362
                  BtnPos *data = g_new(BtnPos, 1); // Allocate memory for the structure
00363
                  data -> pos[0] = i;
                                                         // Store row
00364
                  data->pos[1] = j;
                                                          // Store column
00365
                  // Set the structure as data on the button
g_object_set_data(G_OBJECT(btnGrid[i][j]), "button-data", data);
00366
00367
00368
                  g_signal_connect(btnGrid[i][j], "clicked", G_CALLBACK(on_btnGrid_clicked), score_button);
      // Pass score_button as data
00370
                  gtk_grid_attach(GTK_GRID(grid), btnGrid[i][j], j, i, 1, 1);
     // Attach btnGrid to the grid
00371
              }
00372
00373
00374
          // Make the btnGrid expand to fill the available space
00375
          for (int i = 0; i < 3; i++)
00376
00377
              gtk_widget_set_vexpand(btnGrid[i][0], TRUE);
00378
              gtk_widget_set_hexpand(btnGrid[i][0], TRUE);
00379
00380
          gtk_widget_set_vexpand(score_button, TRUE);
00381
          gtk_widget_set_hexpand(score_button, TRUE);
00382
00383
          // Add the box to the window
00384
          gtk container add(GTK CONTAINER(window), box);
00385
00386
          // Show everything
00387
          gtk_widget_show_all(window);
00388
00389
          // Start the GTK main loop
          gtk_main();
00390
00391
00392
          return SUCCESS;
00393 }
00394 /*
00395 ====
00396 END OF MAIN
00397 ==
00398 */
```

6.21 src/minimax.c File Reference

#include <minimax.h>

Functions

• int max (int a, int b)

Returns the maximum of two integers.

int min (int a, int b)

Returns the minimum of two integers.

• struct Position findBestMove (int board[3][3])

Finds the best move for the bot in the Tic-Tac-Toe game.

int minimax (int board[3][3], int depth, bool isMax)

Implements the Minimax algorithm to evaluate the best move for the bot.

int evaluate (int b[3][3])

Evaluates the current board state to determine if there is a winner.

• bool isMovesLeft (int board[3][3])

Checks if there are any moves left on the board.

• int loadBoardStates (struct BoardState boardStates[])

Loads board states and their best moves from a file.

bool checkAndUpdateBestMove (int board[3][3], struct Position *bestMove, struct BoardState boardStates[], int count)

Checks if the current board configuration exists in the lookup table and updates the best move.

void writeBestMoveToFile (int board[3][3], struct Position bestMove)

Appends the current board state and the best move to a file.

Variables

• int depthCounter = 0

6.21.1 Function Documentation

checkAndUpdateBestMove()

```
bool checkAndUpdateBestMove (
    int board[3][3],
    struct Position * bestMove,
    struct BoardState boardStates[],
    int count)
```

Checks if the current board configuration exists in the lookup table and updates the best move.

This function compares the current board with previously saved board states in the boardStates array. If a matching board configuration is found, it updates the provided bestMove structure with the best move associated with that board state. The function returns true if a match is found and the move is updated, and false if no match is found in the lookup table.

Parameters

board	The current Tic Tac Toe board to check against the saved states.
bestMove	A pointer to the Position structure where the best move will be stored if a match is found.
boardStates	An array of BoardState structures containing previously saved board configurations and their
	best moves.
count	The number of saved board states in the boardStates array.

Returns

true if a matching board configuration is found and the best move is updated, false otherwise.

See also

BoardState, Position

Definition at line 419 of file minimax.c.

evaluate()

```
int evaluate ( int b[3][3])
```

Evaluates the current board state to determine if there is a winner.

This function checks the Tic-Tac-Toe board for winning conditions, i.e., it checks rows, columns, and diagonals for three consecutive marks (either BOT or PLAYER1). It returns a score based on the result:

- +10 if the BOT wins.
- -10 if PLAYER1 wins.
- 0 if there is no winner yet (no winner in rows, columns, or diagonals).

Parameters

b A 3x3 array representing the Tic-Tac-Toe board.

Returns

The evaluation score:

- +10 for a BOT win,
- -10 for a PLAYER1 win,
- 0 if there is no winner.

See also

BOT, PLAYER1

Definition at line 220 of file minimax.c.

findBestMove()

Finds the best move for the bot in the Tic-Tac-Toe game.

This function first checks if the best move is already stored in memory by looking through previous board states. If the move is found, it is returned. If no best move is found in memory, it traverses all the empty cells on the board, evaluates the potential moves using the minimax algorithm, and returns the optimal move.

Parameters

board A 3x3 array representing the current Tic-Tac-Toe board.

Returns

The best move for the bot as a struct Position containing the row and column.

See also

minimax, loadBoardStates, checkAndUpdateBestMove, writeBestMoveToFile

Definition at line 71 of file minimax.c.

isMovesLeft()

```
bool isMovesLeft (
          int board[3][3])
```

Checks if there are any moves left on the board.

This function determines if there are any empty cells left on a 3x3 board. It uses one of the following implementations based on compilation options:

- A standard C implementation when DISABLE_ASM is defined.
- Optimized assembly instructions for either AArch64 or x86 platforms.

The choice of implementation depends on the platform and preprocessor directives.

Parameters

board A 3x3 array representing the board state. Each cell should contain:

- EMPTY (typically 0) if the cell is empty.
- Any non-zero value if the cell is occupied.

Returns

true if there are empty cells; otherwise, false.

Note

If DISABLE ASM is defined, the function uses pure C logic.

For AArch64 platforms, the function uses assembly instructions for efficient scanning.

For x86 platforms, the function uses assembly instructions for efficient scanning.

Example usage:

```
int board[3][3] = {
     {1, 2, 0},
     {0, 1, 2},
     {2, 1, 0}
};
bool movesLeft = isMovesLeft(board);
// movesLeft will be true as there are empty cells (0s).
```

Definition at line 270 of file minimax.c.

loadBoardStates()

Loads board states and their best moves from a file.

This function attempts to open a file containing saved board states and the corresponding best move for each state. If the file does not exist, a new file is created. It reads the board configurations and the best move for each board, storing them in the provided boardStates array.

Each line in the file represents one board state. The board is stored as a 3x3 grid, where 'x' denotes the BOT's move, 'o' denotes PLAYER1's move, and empty spaces are represented as '' (empty). The best move for each board is also saved in the file.

Parameters

boardStates	An array of BoardState structures to store the loaded board states.

Returns

The number of boards loaded from the file. If the file does not exist, it returns 0 and creates a new file.

See also

BoardState, FILE_BESTMOV

Definition at line 366 of file minimax.c.

max()

Returns the maximum of two integers.

This function computes the maximum of two integers using either:

- A standard C implementation when DISABLE_ASM is defined.
- · Optimized assembly instructions for either AArch64 or x86 platforms.

The choice of implementation depends on the platform and preprocessor directives.

Parameters

а	The first integer to compare.
b	The second integer to compare.

Returns

The greater of the two integers.

Note

If ${\tt DISABLE_ASM}$ is defined, the function uses pure C logic.

For AArch64 platforms, the function uses assembly instructions with conditional selection.

For x86 platforms, the function uses assembly instructions with register manipulation and branching.

Example usage:

```
int result = max(10, 20);
// result now holds the value 20.
Definition at line 5 of file minimax.c.
```

min()

```
int min (
     int a,
     int b)
```

Returns the minimum of two integers.

This function computes the minimum of two integers using either:

- A standard C implementation when DISABLE_ASM is defined.
- · Optimized assembly instructions for either AArch64 or x86 platforms.

The choice of implementation depends on the platform and preprocessor directives.

Parameters

а	The first integer to compare.
b	The second integer to compare.

Returns

The smaller of the two integers.

Note

If DISABLE_ASM is defined, the function uses pure C logic.

For AArch64 platforms, the function uses assembly instructions with conditional selection.

For x86 platforms, the function uses assembly instructions with register manipulation and branching.

Example usage:

```
int result = min(10, 20);
// result now holds the value 10.
Definition at line 38 of file minimax.c.
```

minimax()

```
int minimax (
    int board[3][3],
    int depth,
    bool isMax)
```

Implements the Minimax algorithm to evaluate the best move for the bot.

The function recursively evaluates all possible moves using the Minimax algorithm. It returns the best score for the current player (maximizer or minimizer) based on the game state. The algorithm chooses the optimal move for the bot and evaluates the game state at each depth. The depth is capped if Minimax Godmode is not enabled. If there are no moves left or the game is over, it returns the evaluation score.

Parameters

board	A 3x3 array representing the current Tic-Tac-Toe board.
depth	The current depth in the game tree.
isMax	Boolean flag indicating whether it is the maximizer's turn (bot) or the minimizer's turn (player).

Returns

The best score for the current move based on the evaluation function.

See also

```
evaluate, isMovesLeft, max, min
```

Definition at line 136 of file minimax.c.

writeBestMoveToFile()

```
void writeBestMoveToFile (
          int board[3][3],
          struct Position bestMove)
```

Appends the current board state and the best move to a file.

This function writes the current Tic Tac Toe board state to a file, encoding the board as a sequence of characters where 'o' represents Player 1, 'x' represents the Bot, and 'b' represents an empty cell. After writing the board state, it appends the best move (row and column) for the current board to the same file.

Parameters

board	The current Tic Tac Toe board to write to the file.
bestMove	The best move to be made, represented by its row and column indices.

See also

Position, BoardState

Definition at line 436 of file minimax.c.

6.21.2 Variable Documentation

depthCounter

int depthCounter = 0
Definition at line 3 of file minimax.c.

6.22 minimax.c

Go to the documentation of this file.

```
00001 #include <minimax.h>
00002
00003 int depthCounter = 0;
00004
00005 int max(int a, int b)
00006 {
00007 #if (DISABLE_ASM)
80000
          return (a > b) ? a : b;
00009 #else
00010
         int result;
00011 #ifdef __aarch64__
00012
        __asm__(
                                       // Move 'a' to result
00013
               "mov %w0, %w1;"
00014
               "cmp w0, w2;" // Compare result (a) and b "csel w0, w2, w2, ge;" // If a >= b, keep a in result; otherwise, move b to result
               "cmp %w0, %w2;"
00015
                                     // Output
// Inputs
               : "=&r" (result)
00016
               : "r" (a), "r" (b)
00017
               : "cc"
00018
                                        // Clobbered flags (condition codes)
00018 : 66
00019 );
00020 #else //x86-64
        00021
00022
00023
               "cmpl %%ebx, %%eax;" // Compare eax and ebx
00024
00025
                                      // If a >= b, jump to label 1
               "jge 1f;"
               "movl %%ebx, %%eax;" // Otherwise, move ebx to eax "1:;"
00026
00027
               "movl %%eax, %0;" // Move result back to C variable
: "=r"(result) // Output
: "r"(a), "r"(b) // Inputs
: "%eax", "%ebx" // Clobbered registers
00028
00029
00030
00031
00032
00033 #endif
00034
          return result:
00035 #endif
00036 }
00037
00038 int min(int a, int b)
00039 {
00040 #if (DISABLE_ASM)
          return (a < b) ? a : b;
00041
00042 #else
00043
        int result;
00044 #ifdef __aarch64__
00045
              "mov %w0, %w1;"
                                       // Move 'a' to result
00046
               "cmp %w0, %w2;" // Compare result (a) and b
"csel %w0, %w0, %w2, le;" // If a <= b, keep a in result; otherwise, move b to result
00047
00048
                                     // Output
               : "=&r" (result)
00049
               : "r" (a), "r" (b)
                                       // Inputs
00050
00051
               : "cc"
                                        // Clobbered flags (condition codes)
00052 );
00053 #else //x86-64
00054
        ___asm___(
               "movl %1, %%eax;" // Move 'a' to eax
"movl %2, %%ebx;" // Move 'b' to ebx
00055
               00057
00058
00059
               "1:;"
00060
00061
               "movl %%eax, %0;" // Move result back to C variable
               : "=r"(result) // Output
: "r"(a), "r"(b) // Inputs
: "%eax", "%ebx" // Clobbered registers
00062
00063
00064
         );
00065
00066 #endif
00067
         return result:
00068 #endif
00069 }
00070
00071 struct Position findBestMove(int board[3][3])
00072 {
00073
           int bestVal = -1000;
          struct Position bestMove;
00074
```

6.22 minimax.c 57

```
00076
          struct BoardState boardStates[MAX_BOARDS];
00077
00078 #if !(DISABLE LOOKUP)
00079
          startElapseTime();
int boardCount = loadBoardStates(boardStates);
08000
          stopElapseTime("Loading lookup table");
00082 #endif
00083
00084
          bestMove.row = ERROR;
          bestMove.col = ERROR;
00085
00086
00087
          startElapseTime();
00088 #if !(DISABLE_LOOKUP)
00089
          if (checkAndUpdateBestMove(board, &bestMove, boardStates, boardCount))
00090
              stopElapseTime("Find best move in lookup table");
00091
00092
               PRINT\_DEBUG ("Best move found in memory: Row = %d, Col = %d\n", bestMove.row, bestMove.col); 
00093
         }
00094
          else
00095 #endif
00096
00097
              startElapseTime();
00098
              // Traverse all cells, evaluate minimax function for
00099
              // all empty cells. And return the cell with optimal
00100
              // value.
00101
              for (int i = 0; i < 3; i++)
00102
                   for (int j = 0; j < 3; j++)
00103
00104
                   {
                       // Check if cell is empty
if (board[i][j] == EMPTY)
00105
00106
00107
00108
                           // Make the move
00109
                           board[i][j] = BOT;
00110
00111
                           // compute evaluation function for this
                           // move.
00112
00113
                           int moveVal = minimax(board, 0, false);
00114
                           PRINT_DEBUG("[DEBUG] Depth exited at -> %d\n", depthCounter);
00115
                           // Undo the move
00116
                           board[i][j] = EMPTY;
00117
00118
                           // If the value of the current move is more than the best value, then update best
00119
                           if (moveVal > bestVal)
00120
00121
                               bestMove.row = i;
                               bestMove.col = j;
00122
00123
                               bestVal = moveVal;
00124
                           }
00125
00126
                  }
00127
              stopElapseTime("Minimax depth search");
00128
00129
              writeBestMoveToFile(board, bestMove);
00130
00131
00132
          depthCounter = 0;
00133
          return bestMove;
00134 }
00135
00136 int minimax(int board[3][3], int depth, bool isMax)
00137 {
00138 #if DEBUG
00139
          depthCounter++;
00140 #endif
00141
          int score = evaluate(board);
00142
          // If Maximizer has won the game return his/her
00143
          // evaluated score
00144
          if (score == 10)
00145
              return score;
00146
          // If Minimizer has won the game return his/her
00147
00148
          // evaluated score
00149
          if (score == -10)
00150
              return score;
00151
          \ensuremath{//} If there are no more moves and no winner then
00152
          // it is a tie
00153
          if (isMovesLeft(board) == false)
00154
00155
              return 0;
00156
00157 #if !(MINIMAX_GODMODE)
       if (depth > 2)
00158
              return 0;
00159
00160 #endif
```

```
00162
          // If this maximizer's move
00163
          if (isMax)
00164
              int best = -1000:
00165
00166
              // Traverse all cells
00167
00168
               for (int i = 0; i < 3; i++)
00169
                   for (int j = 0; j < 3; j++)
00170
00171
                       // Check if cell is empty
00172
                       if (board[i][j] == EMPTY)
00173
00174
00175
                            // Make the move
00176
                           board[i][j] = BOT;
00177
00178
                           // Call minimax recursively and choose
                            // the maximum value
00180
                           best = max(best, minimax(board, depth + 1, !isMax));
00181
00182
                            // Undo the move
                           board[i][j] = EMPTY;
00183
00184
00185
                  }
00186
00187
              return best;
00188
00189
          // If this minimizer's move
00190
00191
          else
00192
00193
              int best = 1000;
00194
              // Traverse all cells
for (int i = 0; i < 3; i++)</pre>
00195
00196
00197
                   for (int j = 0; j < 3; j++)
00199
00200
                       // Check if cell is empty
00201
                       if (board[i][j] == EMPTY)
00202
                            // Make the move
00203
                           board[i][j] = PLAYER1;
00204
00205
00206
                           // Call minimax recursively and choose
00207
                            // the minimum value
00208
                           best = min(best,
00209
                                       minimax(board, depth + 1, !isMax));
00210
00211
                            // Undo the move
00212
                           board[i][j] = EMPTY;
00213
00214
                  }
00215
00216
              return best;
00218 }
00219
00220 int evaluate(int b[3][3])
00221 {
          // Checking for Rows for X or O victory.
00222
00223
          for (int row = 0; row < 3; row++)</pre>
00224
00225
               if (b[row][0] == b[row][1] &&
00226
                   b[row][1] == b[row][2])
00227
00228
                   if (b[row][0] == BOT)
00229
                      return +10;
00230
                   else if (b[row][0] == PLAYER1)
00231
                      return -10;
00232
00233
          }
00234
00235
          // Checking for Columns for X or O victory.
00236
          for (int col = 0; col < 3; col++)</pre>
00237
              if (b[0][col] == b[1][col] &&
   b[1][col] == b[2][col])
00238
00239
00240
              {
00241
                   if (b[0][col] == BOT)
00242
                       return +10;
00243
00244
                   else if (b[0][col] == PLAYER1)
00245
                      return -10;
00246
              }
00247
          }
```

6.22 minimax.c 59

```
00248
00249
          // Checking for Diagonals for X or O victory.
00250
          if (b[0][0] == b[1][1] && b[1][1] == b[2][2])
00251
               if (b[0][0] == BOT)
00252
00253
                   return +10;
               else if (b[0][0] == PLAYER1)
00255
                  return -10;
00256
          }
00257
          if (b[0][2] == b[1][1] && b[1][1] == b[2][0])
00258
00259
               if (b[0][2] == BOT)
00260
                   return +10;
00261
00262
               else if (b[0][2] == PLAYER1)
00263
                  return -10;
00264
00265
00266
          // Else if none of them have won then return 0
00267
          return 0;
00268 }
00269
00270 bool isMovesLeft(int board[3][3])
00271 {
00272 #if (DISABLE_ASM)
00273
        for (int i = 0; i<3; i++)
00274
              for (int j = 0; j < 3; j++)
                 if (board[i][j] == EMPTY)
00275
00276
                       return true;
          return false;
00277
00278
00279 #else
00280
          int result;
00281 #ifdef __aarch64__
00282 <u>asm</u> (
               "mov x1, #0;"
                                         // x1 = i = 0
00283
00284
               "outer_loop:;"
               "cmp x1, #3;"
                                        // if i >= 3, go to return_false
00286
               "bge return_false;"
00287
00288
               "mov x2, #0;"
                                        // x2 = j = 0
               "inner_loop:;"
00289
               "cmp x2, #3;"
                                        // if j >= 3, increment i
00290
00291
               "bge increment_i;"
00292
               // Calculate board[i][j]
00293
                                         // Copy i to x3
00294
               "mov x3, x1;"
               "lsl x4, x1, #3;"
               "lsl x4, x1, #3;" // x4 = i * 8
"add x3, x4, x1, lsl #2;" // x3 = i * 8 + i * 4 = i * 12
"add x3, %1, x3;" // x3 = board + (i * 12), points to board[i]
"ldr w0, [x3, x2, lsl #2];" // Load board[i][j] (each int is 4 bytes)
00295
00296
00297
00298
00299
               "cbz w0, return_true;" // If board[i][j] == 0, go to return_true
00300
00301
               "add x2, x2, #1;"
00302
                                       // Increment i
00303
               "b inner_loop;"
00304
00305
               "increment_i:;"
00306
               "add x1, x1, #1;"
                                       // Increment i
00307
               "b outer_loop;"
00308
               "return_false:;"
00309
               "mov %w0, #0;
00310
                                        // Set result to 0 (false)
00311
00312
00313
               "return_true:;"
                                        // Set result to 1 (true)
               "mov %w0, #1;"
00314
00315
00316
               "end:;"
               : "=r"(result)
                                         // Output operand
               00318
00319
00320
          );
00321 #else
              //x86-64
00322
          __asm__(
00323
               "xor %%rbx, %%rbx;" // rbx = i = 0
00324
               "outer_loop:;"
              "cmp $3, %%ebx;" // if i >= 3, return false "jge return_false;"
00325
00326
00327
               "xor %%rcx, %%rcx;" // rcx = j = 0
00328
00329
               "inner_loop:;"
               "cmp $3, %%ecx;" // if j >= 3, increment i "jge increment_i;"
00330
00331
00332
               // Calculate board[i][j]
00333
00334
               "mov %%rbx, %%rdx;"
                                                 // Copy i to rdx
```

```
// rdx = i * 12 (calculate row offset)
               "imul $12, %%rdx, %%rdx;"
               "add %1, %%rdx;" // rdx = board + (i * 12), points to board[i]
"mov (%%rdx, %%rcx, 4), %%eax;" // Load board[i][j]
00336
00337
00338
               "test %%eax, %%eax;" // Check if board[i][j] == 0
"jz return_true;" // If board[i][j] == 0, return true
00339
00340
00341
00342
               "inc %%rcx;" // Increment j
00343
               "jmp inner_loop;"
00344
               "increment_i:;"
00345
               "inc %%rbx;" // Increment i
00346
00347
               "jmp outer_loop;"
00348
00349
               "return_false:;"
               "mov $0, %0;" // Set result to 0 (false)
"jmp end;"
00350
00351
00352
00353
               "return_true:;"
00354
               "mov $1, %0;" // Set result to 1 (true)
00355
               "end:;"
00356
               : "=r" (result)
00357
                                               // Output operand
               : "r"(board)
00358
                                               // Input operand
00359
               : "rbx", "rcx", "rdx", "rax" // Clobbered registers
00360
00361 #endif
00362
          return result;
00363 #endif
00364 }
00365
00366 int loadBoardStates(struct BoardState boardStates[])
00367 {
00368
          FILE *file = fopen(FILE_BESTMOV, "r");
00369
           if (file == NULL)
00370
               PRINT_DEBUG("%s <- File does not exist. Creating new file.\n", FILE_BESTMOV); FILE \starfile = fopen(FILE_BESTMOV, "w");
00371
00373
               PRINT_DEBUG("Text file created.\n");
00374
               fclose(file);
00375
               return 0; // No boards loaded
00376
00377
          PRINT DEBUG("File exist. Checking.\n");
00378
           int count = 0;
           char line[100];
00379
00380
           while (fgets(line, sizeof(line), file) != NULL && count < MAX_BOARDS)</pre>
00381
               // Parse the line
00382
               char *token = strtok(line, ",");
00383
               int index = 0;
00384
00385
00386
               // Read the board condition
00387
               while (token != NULL && index < 9)
00388
                    if (strcmp(token, "x") == 0)
00389
00390
                   {
00391
                        boardStates[count].board[index / 3][index % 3] = BOT;
00392
00393
                    else if (strcmp(token, "o") == 0)
00394
                        boardStates[count].board[index / 3][index % 3] = PLAYER1;
00395
00396
00397
                   else
00398
                   {
00399
                        boardStates[count].board[index / 3][index % 3] = EMPTY;
00400
00401
                   token = strtok(NULL, ",");
00402
                   index++;
00403
               }
00404
00405
               // Read the best move
00406
               if (token != NULL)
00407
               {
00408
                   boardStates[count].bestMove.row = atoi(token);
                   token = strtok(NULL, ",");
boardStates[count].bestMove.col = atoi(token);
00409
00410
00411
00412
               count++;
00413
           }
00414
00415
          fclose(file);
00416
           return count; // Return the number of boards loaded
00417 }
00418
00419 bool checkAndUpdateBestMove(int board[3][3], struct Position *bestMove, struct BoardState
      boardStates[], int count)
00420 {
```

```
00421
                       for (int i = 0; i < count; i++)
00423
                                if (memcmp(board, boardStates[i].board, sizeof(boardStates[i].board)) == 0)
00424
                                {
00425
                                         // Board matches, update the best move
                                         *bestMove = boardStates[i].bestMove;
00426
                                         PRINT_DEBUG("Found position in lookup table\n");
00427
00428
                                         PRINT_DEBUG("Best Move = R:%d C:%d\n", bestMove->row, bestMove->col);
00429
                                         return bestMove; // Board matches, return the best move
00430
00431
                       PRINT DEBUG("Position not found in lookup table\n");
00432
00433
                       return false; // No matching board found
00434 }
00435
00436 void writeBestMoveToFile(int board[3][3], struct Position bestMove)
00437 {
                       FILE *file = fopen(FILE BESTMOV, "a"); // Open the file for appending
00438
                       if (file == NULL)
00439
00440
00441
                                PRINT_DEBUG("Error opening file for writing. -> %s\n", FILE_BESTMOV);
00442
00443
                      }
00444
00445
                       // Write the board state to the file
                       for (int j = 0; j < 3; j++)
00447
00448
                                for (int k = 0; k < 3; k++)
00449
00450
                                         if (board[j][k] == PLAYER1)
00451
                                         {
                                                  fprintf(file, "o,");
PRINT_DEBUG("o,");
00452
00453
00454
00455
                                         else if (board[j][k] == BOT)
00456
00457
                                                  fprintf(file, "x,");
                                                  PRINT_DEBUG("x,");
00458
00459
00460
                                         else
00461
                                                  fprintf(file, "b,");
00462
00463
                                                  PRINT_DEBUG("b,");
00464
                                         }
00465
                               }
00466
00467
                       \ensuremath{//} Write the best move to the file
                       fprintf(file, \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ bestMove.row, \ bestMove.col);
00468
                       if (fprintf(file, "%d,%d\n", bestMove.row, bestMove.col) < 0)
00469
00470
00471
                                PRINT_DEBUG("Error writing best move to file. -> %s\n", FILE_BESTMOV);
00472
00473
                        PRINT\_DEBUG("\nAttempting to write best move to file: Row = %d, Col = %d\n", bestMove.row, Col = %d
            bestMove.col);
    PRINT_DEBUG("New best move written to file.\n");
00474
00475
                       fclose(file);
```

6.23 src/ml-naive-bayes.c File Reference

```
#include <ml-naive-bayes.h>
#include <math.h>
```

Functions

• int assignMoveIndex (char move)

Assigns an index to each move ("x", "o", or "b").

void calculateProbabilities (int dataset_size)

Calculates the probabilities for each class and conditional probabilities with Laplace smoothing.

int predictOutcome (struct Dataset board)

Predicts the outcome of a given Tic Tac Toe board based on previously calculated probabilities.

struct Position getBestPosition (int grid[3][3], char player)

Determines the best position for the bot to make a move based on the highest probability.

void resetTrainingData ()

Resets the training data and associated statistics for a fresh training cycle.

• int initData ()

Initializes the training data and model statistics.

void assignCMValue (int actual, int predicted)

Updates the confusion matrix based on actual and predicted outcomes.

void calcConfusionMatrix ()

Calculates the confusion matrix and error probability for the testing dataset.

int getTruthValue (char *str1)

Returns an integer value representing a truth value based on input.

• void calcTrainErrors ()

Calculates the training errors and the probability of error.

void debugDataset (struct Dataset *data, int len)

Debug function to display dataset contents.

Variables

• int positive count = 0

Counter for the number of positive outcomes in the training dataset.

• int negative count = 0

Counter for the number of negative outcomes in the training dataset.

• int cM [4] = {0, 0, 0, 0}

Confusion matrix for evaluating model performance.

• int positiveMoveCount [3][3][3]

3D array to count occurrences of each move for positive outcomes.

• int negativeMoveCount [3][3][3]

3D array to count occurrences of each move for negative outcomes.

• int test PredictedErrors = 0

Counter for the number of errors in the testing dataset predictions.

• int train_PredictedErrors = 0

Counter for the number of errors in the training dataset predictions.

· int predicted

The predicted outcome for the current dataset (1 for positive, 0 for negative).

int actual

The actual outcome for the current dataset (1 for positive, 0 for negative).

· double positiveClassProbability

Probability of a positive outcome in the dataset.

· double negativeClassProbability

Probability of a negative outcome in the dataset.

double probabilityErrors

Probability of error in the predictions, calculated from the testing dataset.

6.23.1 Function Documentation

assignCMValue()

```
void assignCMValue (
    int actual,
    int predicted)
```

Updates the confusion matrix based on actual and predicted outcomes.

This function updates the confusion matrix counters for true positives, false negatives, false positives, and true negatives. It checks the actual and predicted outcomes and increments the appropriate counter in the confusion matrix.

If either the actual or predicted value is ERROR, an error is logged.

Parameters

actual	The actual outcome value (1 for positive, 0 for negative).
predicted	The predicted outcome value (1 for positive, 0 for negative).

See also

cM, ERROR

Definition at line 340 of file ml-naive-bayes.c.

assignMoveIndex()

Assigns an index to each move ("x", "o", or "b").

This function maps the board move characters to their corresponding integer values:

- · 'x' is mapped to the BOT.
- · 'o' is mapped to PLAYER1.
- 'b' is mapped to EMPTY. If the character does not match any of the valid moves, -1 is returned.

Parameters

	move	The character representing the move ('x', 'o', or 'b').	
--	------	---------------------------------------------------------	--

Returns

int The integer corresponding to the move:

- . BOT for 'x',
- PLAYER1 for 'o',
- EMPTY for 'b',
- · ERROR for invalid input.

See also

```
BOT, PLAYER1, EMPTY, ERROR
```

Definition at line 21 of file ml-naive-bayes.c.

calcConfusionMatrix()

```
void calcConfusionMatrix ()
```

Calculates the confusion matrix and error probability for the testing dataset.

This function evaluates the model's performance by calculating the confusion matrix based on actual and predicted outcomes. It iterates through the testing data, compares actual outcomes with predicted ones, and updates the confusion matrix values. The number of prediction errors and the probability of error are also computed.

See also

cM, test_PredictedErrors, probabilityErrors, getTruthValue, predictOutcome

Definition at line 373 of file ml-naive-bayes.c.

calcTrainErrors()

```
void calcTrainErrors ()
```

Calculates the training errors and the probability of error.

This function evaluates the model's performance on the training dataset by comparing predicted outcomes with actual ones. It updates the count of prediction errors and computes the probability of error based on the number of errors and the size of the training dataset.

See also

train PredictedErrors, probabilityErrors, getTruthValue, predictOutcome

Definition at line 424 of file ml-naive-bayes.c.

calculateProbabilities()

Calculates the probabilities for each class and conditional probabilities with Laplace smoothing.

This function calculates:

- · The class probabilities for positive and negative outcomes.
- The conditional probabilities for each move ('x', 'o', 'b') at each position on the board, given the class (positive or negative) with Laplace smoothing applied.

The Laplace smoothing is used to prevent zero probabilities for moves that may not have been observed in the training data. The resulting probabilities are printed for debugging purposes.

Parameters

	dataset_size	The total number of samples in the dataset used for probability calculation.
--	--------------	------------------------------------------------------------------------------

See also

positive_count, negative_count, positiveMoveCount, negativeMoveCount

Definition at line 36 of file ml-naive-bayes.c.

debugDataset()

Debug function to display dataset contents.

This function prints the details of the provided dataset, including the grid values and the corresponding outcomes. It is primarily used for debugging purposes and is not currently in use within the code.

Parameters

data	Pointer to the dataset to be printed.
len	The length of the dataset (number of entries).

See also

```
PRINT_DEBUG
```

Definition at line 451 of file ml-naive-bayes.c.

getBestPosition()

Determines the best position for the bot to make a move based on the highest probability.

This function evaluates all empty positions on the Tic Tac Toe grid and calculates the probability of the bot winning (either as 'x' or 'o') using the pre-calculated move probabilities from the training data. The bot chooses the position with the highest probability of winning, where the move is either 'x' or 'o' depending on the current player. It returns the best position for the bot to make its move.

Parameters

grid	The current state of the Tic Tac Toe game board.
player	The current player, either 'x' or 'o'.

Returns

A struct Position representing the row and column of the best move for the bot. If no valid move is found, it returns an error indicator.

See also

positive_count, negative_count, positiveMoveCount, negativeMoveCount

Definition at line 156 of file ml-naive-bayes.c.

getTruthValue()

Returns an integer value representing a truth value based on input.

This function converts a string input into a corresponding integer truth value. The input string is compared against "positive" and "negative" to determine the output:

- · Returns 1 for "positive".
- · Returns 0 for "negative".
- Returns -1 for any other input, and logs an error message for invalid cases.

Parameters

ĺ	str1	A pointer to the input string to be evaluated.

Returns

int The corresponding truth value:

- 1 for "positive".
- 0 for "negative".
- −1 for invalid inputs.

See also

PRINT_DEBUG

Definition at line 406 of file ml-naive-bayes.c.

initData()

```
int initData ()
```

Initializes the training data and model statistics.

This function resets the training data, then retrieves the training dataset for model training. It processes the dataset to count occurrences of positive and negative outcomes and updates the move counts for each grid position based on the data. Afterward, it calculates training errors and updates the confusion matrix.

If the initial dataset is empty, it attempts to load the data again.

See also

resetTrainingData, getTrainingData, calcTrainErrors, calcConfusionMatrix

Definition at line 273 of file ml-naive-bayes.c.

predictOutcome()

Predicts the outcome of a given Tic Tac Toe board based on previously calculated probabilities.

This function calculates the probabilities of a positive (Player 1 wins) or negative (Bot wins) outcome for a given board state by multiplying the conditional probabilities of each move in the grid with the class probabilities. The prediction is made based on which outcome (positive or negative) has the higher probability.

If the calculated probabilities are zero, indicating that the outcome cannot be predicted with the available data, the function returns -1.

Parameters

board The current Tic Tac Toe board whose outcome needs to be predicted.

Returns

1 if the predicted outcome is positive (Player 1 wins), 0 if negative (Bot wins), and -1 if the outcome cannot be predicted.

See also

positiveClassProbability, negativeClassProbability, positiveMoveCount, negativeMoveCount, assignMoveIndex

Definition at line 88 of file ml-naive-bayes.c.

resetTrainingData()

```
void resetTrainingData ()
```

Resets the training data and associated statistics for a fresh training cycle.

This function resets all relevant variables used in the machine learning model's training process. It clears the outcome counts, resets the move count arrays for each grid position, and reinitializes the confusion matrix. Additionally, it clears the prediction error counters, ensuring that the model starts with a clean state.

See also

positive_count, negative_count, positiveMoveCount, negativeMoveCount, cM, test_PredictedErrors, train_PredictedErrors

Definition at line 247 of file ml-naive-bayes.c.

6.23.2 Variable Documentation

actual

```
int actual
```

The actual outcome for the current dataset (1 for positive, 0 for negative). Definition at line 15 of file ml-naive-bayes.c.

сМ

```
int cM[4] = \{0, 0, 0, 0\}
```

Confusion matrix for evaluating model performance.

Definition at line 7 of file ml-naive-bayes.c.

negative_count

```
int negative_count = 0
```

Counter for the number of negative outcomes in the training dataset.

Definition at line 5 of file ml-naive-bayes.c.

negativeClassProbability

```
double negativeClassProbability
```

Probability of a negative outcome in the dataset.

Definition at line 18 of file ml-naive-bayes.c.

negativeMoveCount

```
int negativeMoveCount[3][3][3]
```

3D array to count occurrences of each move for negative outcomes.

Definition at line 9 of file ml-naive-bayes.c.

positive_count

```
int positive_count = 0
```

Counter for the number of positive outcomes in the training dataset.

Definition at line 4 of file ml-naive-bayes.c.

positiveClassProbability

```
double positiveClassProbability
```

Probability of a positive outcome in the dataset.

Definition at line 17 of file ml-naive-bayes.c.

positiveMoveCount

```
int positiveMoveCount[3][3][3]
```

3D array to count occurrences of each move for positive outcomes.

Definition at line 8 of file ml-naive-bayes.c.

predicted

```
int predicted
```

The predicted outcome for the current dataset (1 for positive, 0 for negative).

Definition at line 14 of file ml-naive-bayes.c.

probabilityErrors

```
double probabilityErrors
```

Probability of error in the predictions, calculated from the testing dataset.

Definition at line 19 of file ml-naive-bayes.c.

test PredictedErrors

```
int test_PredictedErrors = 0
```

Counter for the number of errors in the testing dataset predictions.

Definition at line 11 of file ml-naive-bayes.c.

train_PredictedErrors

```
int train_PredictedErrors = 0
```

Counter for the number of errors in the training dataset predictions.

Definition at line 12 of file ml-naive-bayes.c.

6.24 ml-naive-bayes.c

Go to the documentation of this file.

```
00001 #include <ml-naive-bayes.h>
00002 #include <math.h>
00004 int positive_count = 0;
00005 int negative_count = 0;
00006
00007 int cM[4] = \{0, 0, 0, 0\};
00008 int positiveMoveCount[3][3][3];
00009 int negativeMoveCount[3][3][3];
00010
00011 int test_PredictedErrors = 0;
00012 int train_PredictedErrors = 0;
00013
00014 int predicted;
00015 int actual;
00017 double positiveClassProbability;
00018 double negativeClassProbability;
00019 double probabilityErrors;
00020
00021 int assignMoveIndex(char move) //converts char to int value for easier calculation
00022 {
00023
          switch (move)
00024
          case 'x':
00025
00026
             return BOT;
00027
          case 'o':
             return PLAYER1;
00029
          case 'b':
00030
             return EMPTY;
          default:
00031
             return ERROR;
00032
00033
00034 }
00035
00036 void calculateProbabilities(int dataset_size)
00037 {
00038
          // Calculate class probability
          positiveClassProbability = (double)positive_count / dataset_size;
negativeClassProbability = (double)negative_count / dataset_size;
00039
00040
00041
          PRINT_DEBUG("Positive Class Probability: %lf\n", positiveClassProbability);
00042
          {\tt PRINT\_DEBUG("Negative Class Probability: \$lf \ 'n", negative Class Probability);}
00043
          // Calculate conditional probability with laplace smoothing
00044
00045
          int laplace smoothing = 1;
          for (int row = 0; row < 3; row++)
00046
00047
00048
              for (int col = 0; col < 3; col++)</pre>
00049
00050
                  for (int moveIndex = 0; moveIndex < 3; moveIndex++)</pre>
00051
00052
                      char move;
00053
                      if (moveIndex == 0) //convert
00054
00055
                          move = 'x';
00056
00057
                      else if (moveIndex == 1)
00058
                          move = 'o';
00060
00061
                       else
00062
                          move = 'b';
00063
00064
00065
                      double positiveProbability = (double) (positiveMoveCount[row][col][moveIndex] +
00066
      laplace_smoothing) / (positive_count + 3 * laplace_smoothing);
00067
                      double negativeProbability = (double) (negativeMoveCount[row][col][moveIndex] +
      laplace_smoothing) / (negative_count + 3 * laplace_smoothing);
00068
                      if (positive_count == 0)
00069
                           PRINT\_DEBUG("Probability of %c (positive) at grid(%d,%d): No positive outcomes \\ \\ n", \\
      move, row, col);
00071
                          negativeProbability);
```

```
00072
00073
                      else if (negative_count == 0)
00074
                          00075
      positiveProbability);
                          PRINT_DEBUG("Probability of %c (negative) at grid(%d,%d): No negative outcomes \n",
00076
      move, row, col);
00077
00078
00079
                          PRINT_DEBUG("Probability of %c (positive) at grid(%d,%d): %lf\n", move, row, col,
00080
      positiveProbability);
00081
                          PRINT_DEBUG("Probability of %c (negative) at grid(%d,%d): %lf\n", move, row, col,
      negativeProbability);
00082
00083
00084
              }
00085
          }
00086 }
00087
00088 int predictOutcome(struct Dataset board)
00089 {
          double positiveProbability = positiveClassProbability;
double negativeProbability = negativeClassProbability;
00090
00091
00092
00093
          // required as 0*anything = 0
          if (positiveProbability == 0)
00094
00095
00096
              positiveProbability = 1;
00097
00098
          if (negativeProbability == 0)
00099
          {
00100
              negativeProbability = 1;
00101
00102
          //loops through board grid and sums up probability for each grid
00103
00104
          for (int row = 0; row < 3; row++)
00105
00106
              for (int col = 0; col < 3; col++)
00107
00108
                  int moveIndex = assignMoveIndex(board.grid[row][col]);
                  if (moveIndex != -1)
00109
00110
                  {
                      // PRINT_DEBUG("\nPC_%d, NC_%d, pMC_%d,
00111
      nMC_%d",positive_count,negative_count,positiveMoveCount[row][col][moveIndex],negativeMoveCount[row][col][moveIndex]);
00112
                       if (positive_count > 0)
00113
00114
                          positiveProbability *= (double)positiveMoveCount[row][col][moveIndex] /
      (double) positive_count;
00115
00116
00117
                      if (negative_count > 0)
00118
00119
                          negativeProbability *= (double)negativeMoveCount[row][col][moveIndex] /
      (double)negative_count;
00120
00121
00122
              }
00123
          }
00124
          // guard cases if either negativeProbability is unset
00125
00126
          if (positiveProbability == 1)
00127
          {
00128
              positiveProbability = 0;
00129
00130
          if (negativeProbability == 1)
00131
00132
              negativeProbability = 0:
00133
00134
00135
          // Output probabilities for debugging
00136
          // PRINT_DEBUG("\nPositive: %lf, Negative: %lf Probability: \n", positiveProbability,
      negativeProbability);
00137
00138
          //returns a value based on condition
00139
          if (positiveProbability > negativeProbability)
00140
          {
00141
              // PRINT_DEBUG("Predicted Outcome: Positive\n");
00142
              return 1;
00143
          else if (positiveProbability == 0 || negativeProbability == 0)
00144
00145
00146
              // PRINT_DEBUG("Unable to predict outcome based on available data.");
00147
              return -1;
00148
00149
          else
00150
```

```
// PRINT_DEBUG("Predicted Outcome: Negative\n");
00152
              return 0;
00153
          }
00154 }
00155
00156 struct Position getBestPosition(int grid[3][3], char player)
00157 {
00158
          // Determine whether bot is X or O depending on current player
00159
          char bot = (player == 'x' ? 'o' : 'x');
00160
          char bestMove = 'b';
          int bestRow = -1;
00161
          int bestCol = -1;
00162
00163
          double highestProbability = 0.0;
00164
00165
          int bot_count;
00166
          int(*botMoveCount)[3][3];
00167
          // Use positive or negative count for calculating probability depending on whether bot is \boldsymbol{X} or \boldsymbol{O}
00168
          // Note that for the dataset, negative outcome is for X, meaning the position of O in negative
00169
     outcomes are good for the bot playing as O
00170
          if (bot == 'x')
00171
00172
              bot_count = positive_count;
              botMoveCount = positiveMoveCount;
00173
00174
00175
          else
00176
          {
00177
              bot_count = negative_count;
00178
              botMoveCount = negativeMoveCount;
00179
          }
00180
00181
          for (int row = 0; row < 3; row++)</pre>
00182
00183
              for (int col = 0; col < 3; col++)</pre>
00184
                   // If the grid position is empty
00185
00186
                   if (grid[row][col] != EMPTY)
00187
00188
                       continue;
00189
                   }
00190
                   // Calculate probability for X or O to determine best move for bot
00191
                   for (int moveIndex = 0; moveIndex < 2; moveIndex++)</pre>
00192
00193
00194
                       double moveProbability;
00195
00196
                       if (bot == 'x')
00197
                       {
00198
                           // Calculate probability for move 'x'
00199
                           if (bot count > 0)
00200
                           {
00201
                               moveProbability = (double)botMoveCount[row][col][0] / bot_count;
00202
                           }
00203
                           else
00204
00205
                               moveProbability = 0.0;
00206
00207
00208
                       else
00209
                           // Calculate probability for move 'o'
00210
00211
                           if (bot count > 0)
00212
                           {
00213
                               moveProbability = (double)botMoveCount[row][col][1] / bot_count;
00214
00215
                           else
00216
                           {
00217
                               moveProbability = 0.0;
00218
00219
00220
00221
                       // Update best move and position for bot if it has higher probability
00222
                       if (moveProbability > highestProbability)
00223
                           highestProbability = moveProbability;
00224
00225
                           bestMove = bot;
00226
                           bestRow = row;
00227
                           bestCol = col;
00228
00229
                  }
              }
00230
00231
          }
00232
00233
          // Return best position
00234
          if (bestRow != ERROR && bestCol != ERROR)
00235
00236
              grid[bestRowl[bestCol] = bestMove;
```

```
00237
              PRINT_DEBUG("Best move: %c at grid (%d, %d) with probability: %lf\n", bestMove, bestRow,
      bestCol, highestProbability);
00238
               return (struct Position) {bestRow, bestCol};
00239
00240
          else
00241
          {
               PRINT_DEBUG("\nNo valid move found.\n");
00242
00243
               return (struct Position) {ERROR, ERROR}; // Indicate no valid move found
00244
00245 }
00246
00247 void resetTrainingData() {
00248
          // Reset outcome counts
00249
          positive_count = 0;
00250
          negative_count = 0;
00251
           \ensuremath{//} Reset move count arrays for each grid position
00252
          for (int row = 0; row < 3; row++) {
    for (int col = 0; col < 3; col++) {</pre>
00253
00254
00255
                   for (int moveIndex = 0; moveIndex < 3; moveIndex++) {</pre>
00256
                        positiveMoveCount[row][col][moveIndex] = 0;
00257
                        negativeMoveCount[row][col][moveIndex] = 0;
00258
                   }
00259
              }
00260
          }
00261
00262
           \ensuremath{//} Reset the confusion matrix counters
          cM[0] = 0; // True positive
cM[1] = 0; // False negative
00263
00264
          cM[2] = 0; // False positive
00265
          cM[3] = 0; // True negative
00266
00267
00268
           // Reset prediction errors
00269
           test_PredictedErrors = 0;
00270
          train_PredictedErrors = 0;
00271 }
00272
00273 int initData()
00274 {
00275
           resetTrainingData();
00276
          int retVal = SUCCESS;
00277
00278 doGetTrainingData:
00279
          static bool doOnce = false;
           struct Dataset *trainingData = NULL;
                                                    // Initialize pointer
00280
00281
           int len = getTrainingData(&trainingData); // Pass address of pointer
00282
00283
          if (len <= 0)
00284
00285
               retVal = readDataset(RES_PATH "" DATA_PATH, true);
00286
               if (retVal != SUCCESS)
00287
00288
                   return retVal;
00289
               }
00290
00291
               if (doOnce) //prevents potential loopback/deadlock. Edge case tbh.
00292
00293
                   return BAD PARAM:
00294
00295
00296
               doOnce = true:
00297
              goto doGetTrainingData; //loops until training data is set
00298
00299
00300
           //loops through train dataset for ml training
00301
           for (int i = 0; i < len; i++)</pre>
00302
               // Get outcome class count for each position
if (strcmp(trainingData[i].outcome, "positive") == 0)
00303
00304
00305
               {
00306
                   positive_count++;
00307
                    for (int row = 0; row < 3; row++)
00308
00309
                        for (int col = 0; col < 3; col++)
00310
00311
                            int moveIndex = assignMoveIndex(trainingData[i].grid[row][col]);
00312
                            if (moveIndex != -1)
00313
00314
                                 positiveMoveCount[row][col][moveIndex]++;
00315
                            }
00316
00317
                   }
00318
00319
               else if (strcmp(trainingData[i].outcome, "negative") == 0)
00320
00321
                   negative_count++;
00322
                   for (int row = 0; row < 3; row++)
```

```
{
00324
                      for (int col = 0; col < 3; col++)</pre>
00325
                          int moveIndex = assignMoveIndex(trainingData[i].grid[row][col]);
if (moveIndex != -1)
00326
00327
00328
00329
                               negativeMoveCount[row][col][moveIndex]++;
00330
00331
00332
                  }
              }
00333
00334
00335
          calcTrainErrors();
00336
          calcConfusionMatrix();
00337
          return SUCCESS;
00338 }
00339
00340 void assignCMValue(int actual, int predicted)
00341 {
00342
          // PRINT_DEBUG("\nactual_%i, predicted_%i\n",actual,predicted);
00343
00344
          if (actual == ERROR || predicted == ERROR)
00345
          {
              PRINT_DEBUG("ERROR either value is -1. actual: %d predicted: %d", actual, predicted);
00346
00347
          }
00348
00349
          if (actual == 1)
00350
00351
              if (predicted == 1)
00352
              {
00353
                  cM[0] += 1; // True positive
00354
00355
00356
00357
                  cM[1] += 1; // False negative
00358
00359
00360
          else
00361
          {
00362
              if (predicted == 1)
00363
              {
00364
                  cM[2] += 1; // False positive
00365
              }
00366
              else
00367
              {
00368
                  cM[3] += 1; // True negative
00369
00370
          }
00371 }
00372
00373 void calcConfusionMatrix()
00374 {
00375
          //{\tt Tests} ml on test dataset and stores result in a confusion matrix
00376
00377
          struct Dataset *test = NULL:
00378
          int len = getTestingData(&test);
00379
          // PRINT_DEBUG("Test_Data length: %d\n", len);
00380
          //loops through testing dataset
00381
          if (len > 0)
00382
          { // Ensure len is valid before accessing test
00383
              for (int i = 0; i < len; i++)</pre>
00384
              {
00385
                  actual = getTruthValue(test[i].outcome); //converts char* to int for comparison
00386
                  predicted = predictOutcome(test[i]);
00387
00388
                  \ensuremath{//} checks and updates total errors for test dataset
00389
                  if (actual != predicted)
00390
                  {
00391
                      test_PredictedErrors += 1;
00392
00393
00394
                  //sets value based on actual vs predicted
00395
                  assignCMValue(actual, predicted);
00396
              }
00397
          }
00398
00399
          double i = TESTING_DATA_SIZE;
                                                                // assign macro to double as you cant cast
00400
          probability \texttt{Errors} = (1 \ / \ i) \ \star \ \texttt{test\_PredictedErrors}; \ / / \ round \ to \ 2dp? \ not \ in \ spec \ though
00401
          00402
     probabilityErrors);
00403
          PRINT_DEBUG("TP: %d, FN: %d, FP: %d, TN: %d\n", cM[0], cM[1], cM[2], cM[3]);
00404 }
00405
00406 int getTruthValue(char *strl) //returns an integer value based on input
00407 {
```

```
00408
          if (strcmp(str1, "positive") == 0)
00409
          {
00410
              return 1;
00411
          else if (strcmp(str1, "negative") == 0)
00412
00413
00414
              return 0;
00415
00416
          else
00417
              //guard case if inputs are neither "positive" nor "negative"
00418
              PRINT_DEBUG("ERROR: Not truth value: %p", str1);
00419
00420
              return -1;
00421
00422 }
00423
00424 void calcTrainErrors()
00425 {
00426
          struct Dataset *train = NULL;  // Initialize pointer
00427
          int len = getTrainingData(&train); // Pass address of pointer
00428
          // debugDataset(test,len);
00429
00430
          if (len > 0)
          { // Ensure len is valid before accessing test
00431
00432
             for (int i = 0; i < len; i++)</pre>
00434
                  predicted = predictOutcome(train[i]);
00435
                  actual = getTruthValue(train[i].outcome);
00436
                  // PRINT_DEBUG("Actual dataset outcome: s, Dataset outcome: d, Predicted outcome: d",
     test[i].outcome, actual, predicted);
00437
                 // checks and updates total errors for train dataset
00438
                  if (actual != predicted)
00439
00440
                      train_PredictedErrors += 1;
00441
00442
              }
00443
         }
00444
00445
          double i = TRAINING_DATA_SIZE;
                                                                // assign macro to double var as macros cant
00446
         probabilityErrors = (1 / i) * train_PredictedErrors; // round to 2dp? not in spec though
00447
          PRINT_DEBUG("\nFor training dataset: %d errors, %lf probability of error.\n",
00448
      train_PredictedErrors, probabilityErrors);
00449 }
00450
00451 void debugDataset(struct Dataset *data, int len)
00452 {
          PRINT_DEBUG("%d\n", len);
00453
00454
          if (len > 0)
00455
          { // Ensure len is valid before accessing test
00456
              for (int i = 0; i < len; i++)</pre>
00457
              {
                  PRINT_DEBUG("%d ", i);
for (int j = 0; j < 3; j++)
00458
00459
00460
                  {
00461
                       for (int k = 0; k < 3; k++)
00462
00463
                          PRINT_DEBUG("%c,", data->grid[j][k]);
00464
00465
                  PRINT_DEBUG("%s\n", data->outcome);
00466
00467
              }
00468
          }
00469 }
```

Index

actual	macros.h, 15
ml-naive-bayes.c, 66	Dataset, 8
assignCMValue	grid, 8
ml-naive-bayes.c, 62	outcome, 8
ml-naive-bayes.h, 30	DEBUG
assignMoveIndex	macros.h, 16
ml-naive-bayes.c, 63	debugDataset
ml-naive-bayes.h, 30	ml-naive-bayes.c, 64
•	ml-naive-bayes.h, 31
BAD_PARAM	depthCounter
macros.h, 15	minimax.c, 56
bestMove	DISABLE_ASM
BoardState, 7	macros.h, 16
board	DISABLE_ELAPSED
BoardState, 7	macros.h, 16
BoardState, 7	DISABLE_LOOKUP
bestMove, 7	macros.h, 16
board, 7	doBOTmove
BOT	main.c. 42
macros.h, 15	main.h, 19
btnGrid	Documentation/README.md, 9
main.c, 44	
BtnPos, 7	Doxygen Documentation Generation and Viewing In-
pos, 8	structions, 5
ροδ, θ	elapsedTime.c
calcConfusionMatrix	gEndTime, 35
ml-naive-bayes.c, 63	_
ml-naive-bayes.h, 31	gStartTime, 35
calcTrainErrors	gTime, 35
ml-naive-bayes.c, 63	startElapseTime, 35
ml-naive-bayes.h, 31	stopElapseTime, 35
calculateProbabilities	elapsedTime.h
	startElapseTime, 10
ml-naive-bayes.c, 64	stopElapseTime, 10
ml-naive-bayes.h, 31	EMPTY
checkAndUpdateBestMove	macros.h, 16
minimax.c, 51	ERROR
minimax.h, 23	macros.h, 16
chkPlayerWin	evaluate
main.c, 41	minimax.c, 51
main.h, 19	minimax.h, 24
CLASSES	
macros.h, 15	FILE_BESTMOV
ml-naive-bayes.h, 30	minimax.h, 23
clearGrid	findBestMove
main.c, 42	minimax.c, 52
main.h, 19	minimax.h, 24
сМ	
ml-naive-bayes.c, 66	gEndTime
col	elapsedTime.c, 35
Position, 9	getBestPosition
	ml-naive-bayes.c, 64
data	ml-naive-bayes.h, 32
importData.c, 38	getRandomNo
DATA_PATH	importData.c, 36
importData.h, 12	importData.h, 12
DATA SIZE	getTestingData

76 INDEX

importData.c, 36	minimax.h, 25
importData.h, 12	isPlayer1Turn
getTrainingData	main.c, 45
importData.c, 37	iTie_score
importData.h, 13	main.c, 45
getTruthValue	iWinPos
ml-naive-bayes.c, 65	main.c, 45
ml-naive-bayes.h, 32	len_test
grid	importData.c, 38
Dataset, 8	len train
gStartTime	importData.c, 38
elapsedTime.c, 35	loadBoardStates
gTime elapsedTime.c, 35	minimax.c, 53
elapsed lillie.c, 33	minimax.h, 25
header/elapsedTime.h, 9, 11	, 20
header/importData.h, 11, 14	macros.h
header/macros.h, 14, 17	BAD_PARAM, 15
header/main.h, 18, 22	BOT, 15
header/minimax.h, 22, 28	CLASSES, 15
header/ml-naive-bayes.h, 28, 34	DATA_SIZE, 15
• , ,	DEBUG, 16
iBoard	DISABLE_ASM, 16
main.c, 44	DISABLE_ELAPSED, 16
iGameState	DISABLE_LOOKUP, 16
main.c, 45	EMPTY, 16
importData.c	ERROR, 16
data, 38	MINIMAX_GODMODE, 16
getRandomNo, 36	MODE_2P, 16
getTestingData, 36	MODE_ML, 16
getTrainingData, 37	MODE_MM, 16
len_test, 38	PLAY, 17
len_train, 38	PLAYER1, 17
randomNo, 38	PRINT_DEBUG, 17
readDataset, 37	SUCCESS, 17
splitFile, 38	TIE, 17
testingFile, 38	WIN, 17
trainingFile, 38	main
importData.h	main.c, 42
DATA_PATH, 12	main.c
getRandomNo, 12	btnGrid, 44
getTestingData, 12	chkPlayerWin, 41
getTrainingData, 13	clearGrid, 42
readDataset, 13	doBOTmove, 42
RES_PATH, 12	iBoard, 44
splitFile, 13	iGameState, 45
TEST_PATH, 12	iPlayer1_score, 45
TRAIN_PATH, 12	iPlayer2_score, 45
initData	isMLAvail, 45
ml-naive-bayes.c, 65	isPlayer1Turn, 45
ml-naive-bayes.h, 33	iTie_score, 45
iPlayer1_score	iWinPos, 45
main.c, 45	main, 42
iPlayer2_score	on_btnGrid_clicked, 43
main.c, 45	on_btnScore_clicked, 43
isMLAvail	playerMode, 45
main.c, 45	showWin, 44
isMovesLeft	updateScoreBtn, 44
minimax.c, 52	main.h

INDEX 77

chkPlayerWin, 19	negativeClassProbability, 67
clearGrid, 19	negativeMoveCount, 67
doBOTmove, 19	positive count, 67
on_btnGrid_clicked, 20	positiveClassProbability, 67
on_btnScore_clicked, 20	positive Move Count, 67
	•
showWin, 21	predicted, 67
updateScoreBtn, 21	predictOutcome, 66
mainpage.md, 34	probabilityErrors, 67
max	resetTrainingData, 66
minimax.c, 54	test_PredictedErrors, 67
minimax.h, 26	train PredictedErrors, 67
MAX BOARDS	ml-naive-bayes.h
minimax.h, 23	assignCMValue, 30
min	assignMoveIndex, 30
	_
minimax.c, 54	calcConfusionMatrix, 31
minimax.h, 26	calcTrainErrors, 31
minimax	calculateProbabilities, 31
minimax.c, 55	CLASSES, 30
minimax.h, 27	debugDataset, 31
minimax.c	getBestPosition, 32
checkAndUpdateBestMove, 51	getTruthValue, 32
depthCounter, 56	initData, 33
evaluate, 51	predictOutcome, 33
findBestMove, 52	resetTrainingData, 33
	G
isMovesLeft, 52	TESTING_DATA_SIZE, 30
loadBoardStates, 53	TRAINING_DATA_SIZE, 30
max, 54	mode
min, 54	PlayerMode, 9
minimax, 55	MODE_2P
writeBestMoveToFile, 55	macros.h, 16
minimax.h	MODE ML
checkAndUpdateBestMove, 23	macros.h, 16
evaluate, 24	MODE MM
FILE BESTMOV, 23	macros.h, 16
-	macros.n, 16
findBestMove, 24	pogativo count
isMovesLeft, 25	negative_count
loadBoardStates, 25	ml-naive-bayes.c, 67
max, <mark>26</mark>	negativeClassProbability
MAX_BOARDS, 23	ml-naive-bayes.c, 67
min, 26	negativeMoveCount
minimax, 27	ml-naive-bayes.c, 67
printFileContents, 27	
writeBestMoveToFile, 28	on_btnGrid_clicked
MINIMAX_GODMODE	main.c, 43
	main.h, <mark>20</mark>
macros.h, 16	on_btnScore_clicked
ml-naive-bayes.c	main.c, 43
actual, 66	main.h, 20
assignCMValue, 62	
assignMoveIndex, 63	outcome
calcConfusionMatrix, 63	Dataset, 8
calcTrainErrors, 63	5. 0.4
calculateProbabilities, 64	PLAY
cM, 66	macros.h, 17
	PLAYER1
debugDataset, 64	macros.h, 17
getBestPosition, 64	PlayerMode, 8
getTruthValue, 65	mode, 9
initData, 65	txt, 9
negative_count, 67	playerMode
	playorivious

78 INDEX

main.c, 45 pos	test_PredictedErrors ml-naive-bayes.c, 67
BtnPos, 8	TESTING_DATA_SIZE
Position, 9	ml-naive-bayes.h, 30
col, 9	testingFile
row, 9	importData.c, 38
positive_count	TicTacToe, 1
ml-naive-bayes.c, 67	TIE
positiveClassProbability	macros.h, 17
ml-naive-bayes.c, 67	TRAIN_PATH
positiveMoveCount	importData.h, 12
ml-naive-bayes.c, 67	train_PredictedErrors
predicted	ml-naive-bayes.c, 67
ml-naive-bayes.c, 67	TRAINING_DATA_SIZE
predictOutcome	ml-naive-bayes.h, 30
ml-naive-bayes.c, 66	trainingFile
ml-naive-bayes.h, 33	importData.c, 38
PRINT_DEBUG	txt
macros.h, 17	PlayerMode, 9
printFileContents	
minimax.h, 27	updateScoreBtn
probabilityErrors	main.c, 44
ml-naive-bayes.c, 67	main.h, 21
	WIN
randomNo	macros.h, 17
importData.c, 38	writeBestMoveToFile
readDataset	
importData.c, 37	minimax.c, 55
importData.h, 13	minimax.h, 28
RES_PATH	
importData.h, 12	
resetTrainingData	
ml-naive-bayes.c, 66	
ml-naive-bayes.h, 33	
row Desitions 0	
Position, 9	
showWin	
main.c, 44	
main.h, 21	
splitFile	
importData.c, 38	
importData.h, 13	
src/elapsedTime.c, 34, 35	
src/importData.c, 36, 39	
src/main.c, 40, 46	
src/minimax.c, 50, 56	
src/ml-naive-bayes.c, 61, 68	
startElapseTime	
elapsedTime.c, 35	
elapsedTime.h, 10	
stopElapseTime	
elapsedTime.c, 35	
elapsedTime.h, 10	
SUCCESS	
macros.h, 17	
TEST_PATH	
importData.h, 12	