Project 1

Bulls and Cows Game

CIS 17C Section 48942

Name: Marlo Zeroth

Date: 12/01/2015

**Introduction**

Title: Bulls and Cows Game

Bulls and Cows is a code breaking game similar to Mastermind, but was developed earlier. When Bulls and Cows begins the player is presented the option of choosing among easy, medium and hard difficulty. The difficulty determines playing with or without duplicates letters in the puzzle and the length of puzzle word. The puzzle words are isograms. Isograms are words where each letter is repeated only once or if a letter is repeated more than once all letters must repeat by the same amount. A valid puzzle word, for example, is anna which has the letters ‘a’ and ‘n’ repeated twice and is an isogram. Cows are letters that are in the puzzle, but not in the right spot. Bulls are letters that are in the puzzle in the right spot. The player can enter letters from A to Z without regard for case. After the player enters his/her choices the program creates a random code for the player to break. The player can enter up to 8 guesses to win the game. If a letter of the guess is in the puzzle, but not in the right spot the program prints to the screen the number of cows. If the player entered a letter that is in the puzzle in the right position, the number of bulls is printed to the screen. If no letter of the guess are in the puzzle 0 is printed for cows and also 0 for bulls. If the players runs out of turns before the game ends, the player looses. The player is given the choice of starting another game after he/she breaks the code or runs out of guesses.

**Summary**

Project size:

Number of variables: About 47 major variables. The most important where in my classes. I used a lot of the List<T> to create the guessEntries where the user enters and receives information about their guess and code. The most important object was playSpace instantiated from the PlaySpace class.

Number of methods: 69

I implemented most of the concepts we have learned up to stacks, queues, lists, doubly linkeds from the course book and the lectures. The game still needs to be completed. It still needs the ability to save and load games from binary files, to play at the last save point, connect to a database, implement threads and sockets.

The program itself heavily uses a lot of the concepts for linked lists and templates and QWidgets. The guessEntry class is an implementation of Qwidget which itself includes other Qwigets. I implement a pop ups, a dialog, error message, and regular expressions in the logindialog.h file.

**Description**

The main point of the game is how it functions when a player enters a guess and how the program delivers output the player can use to base their next guess.

**Pseudo Code**

Initialize

Get user input for game play choices for difficulty.

Set size of puzzle and guess to size specified by the difficulty given by player.

If the player chooses medium

Generate a puzzle word that is a length of 4 letters with no duplicate letters in the answer

Else if the player choose hard

Generate a puzzle word with duplicates and length of 6 words.

Else

Generate a puzzle word with 3 letters only and no duplicate letters.

Begin game loop

Get the users guess.

Compare guess to puzzle word.

If guess matches puzzle word exactly

Show the puzzle to the screen, the number of guesses, and a message that player won.

Else if guess does not match answer and guesses are less than 8

Print out dummy puzzle answer made up of ?s, print out cows and bulls for guesses,

and print out guess

Go next loop iteration for next guess

Else

Print out puzzle word, print out cows and bulls for guesses, print guesses,

and message for better luck next time.

End game loop and ask user if they want to start a new game.

If user chooses to play again

Start new game

Else

Exit the game loop.

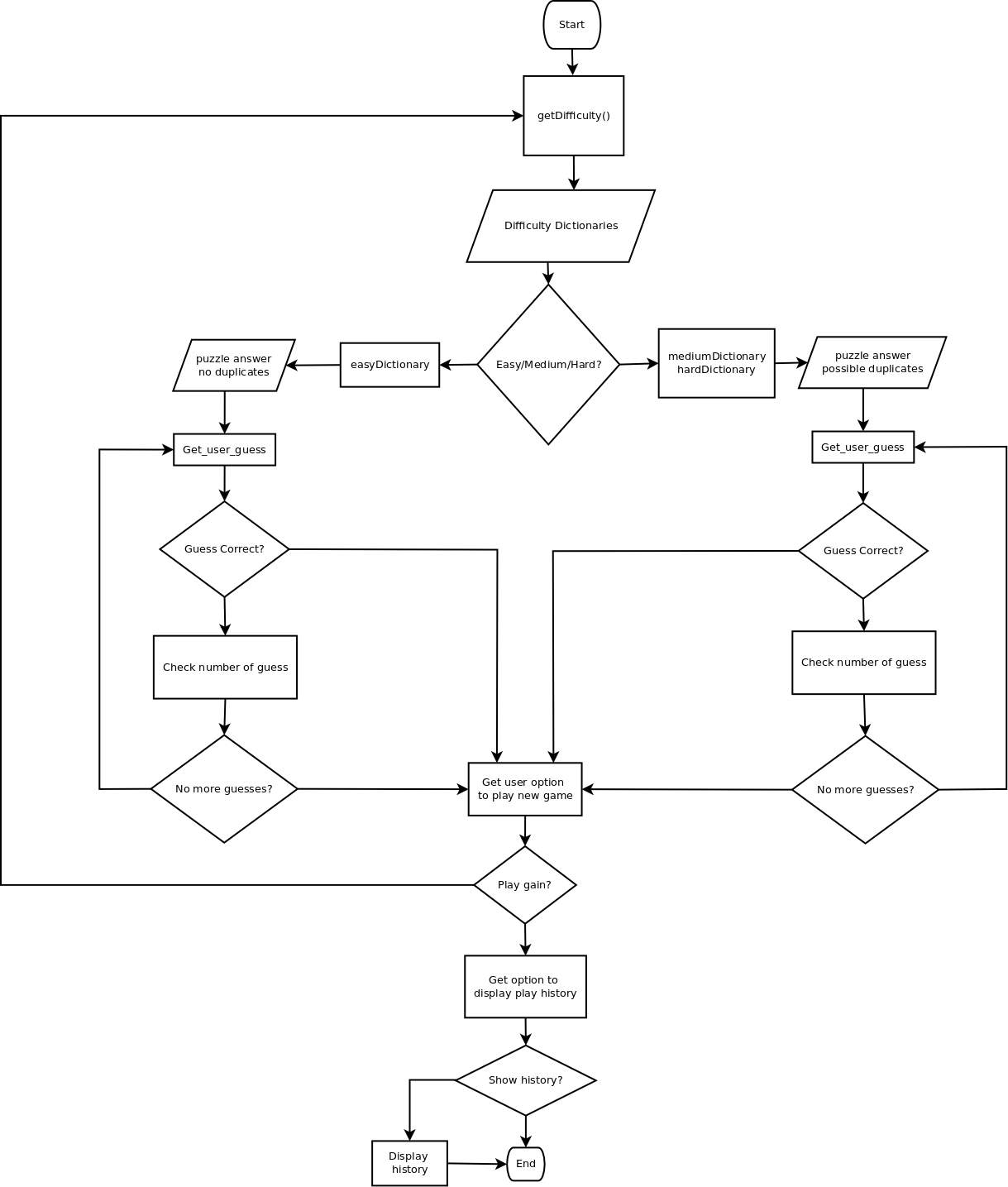
If user chooses to show play history

display play history

else

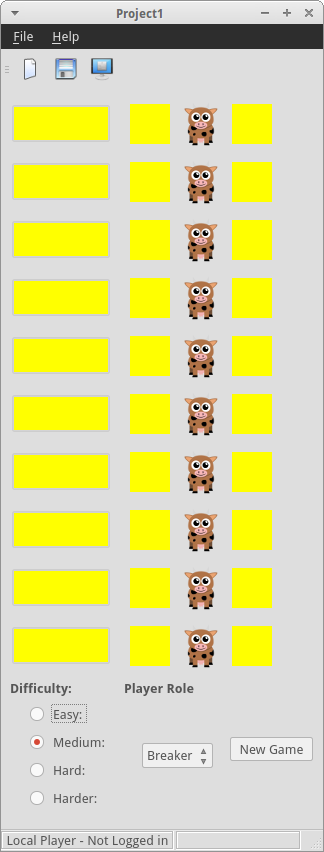
Exit program

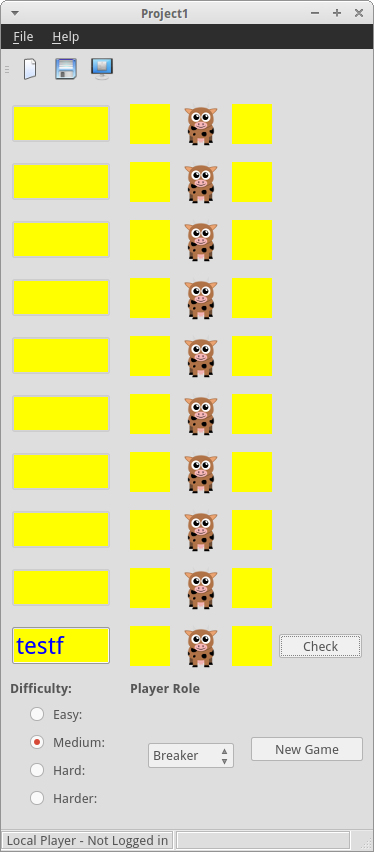
**Flow chart**



**Screen Shots**

Regular program flow.





**Major Variables**

|  |  |  |  |
| --- | --- | --- | --- |
| Type | Variable Name | Description | Location |
| List<T> | guessInterface | Holds guessEntry widgets | playspace.h and playspace.cpp |
| QSet<T> | intersection | Used to get the number of bulls and cows in a guess compared with a code word | Guessentry.h and guessentry.cpp |
| Stack<T> | playHistory | Holds the player guess history | In mainwindow.h not currently used though |
| QString | codeWord | Temporarily holds the code word and checkes it against a users guess | In the guessEntry and playSpace classes. |
| QLabel | bullCount | Temporarily holds and displays the users guess bull count | guessEntry.cpp and guessEntry.h |
| QMainWindow | w | Main object for starting the application | Main.cpp |

**Program**

/\*

\* Emtpy.h

\*

\* Created on: Nov 4, 2015

\* Author: super0

\*/

#ifndef EMPTY\_H\_

#define EMPTY\_H\_

#include <iostream>

#include <cstdlib>

class Empty{

public:

Empty(){

std::cout <<"Structure is empty." << std::endl;

exit (EXIT\_FAILURE);

};

};

#endif /\* EMPTY\_H\_ \*/

#ifndef GUESSENTRY\_H

#define GUESSENTRY\_H

#include <QWidget>

#include <QLabel>

#include <QString>

#include <QPushButton>

#include <QLineEdit>

#include <QPixmap>

#include <QGridLayout>

#include <QRegExp>

#include <QRegExpValidator>

#include <QSet>

#include <QChar>

class GuessEntry : public QWidget

{

Q\_OBJECT

public:

explicit GuessEntry(QPixmap bull, QPixmap cow,

bool current, QWidget \*parent = 0, Qt::WindowFlags f=0);

public slots:

void setCurrent(bool status);

// Sets the current member variable

void setCodeWord(QString code);

// Sets the current code word to test

void reset();

// Resets the displayed widgets

private slots:

void guessFinished(bool);

// Connects to checkButton's clicked(bool) signal and

// emits solved(true) if the codeWord matches the user guess

// or emits finished(true) if the codeWord does not

void enableCheck(const QString &guessWord);

// Enables/disables the checkButton based on whether valid text

// is entered in guessEntry

void setValidation(int test);

// Updates the validation for lineEdit

signals:

void finished(bool);

// Emits if the processing the code and guess words is completed

void solved(bool);

// Emits whether the code word and guess are the same

void current(bool);

// Emits the value of the member variable current

void guessWord(QString,unsigned int, unsigned int);

// Emits the guess word and its bulls and cows

void codeWordChanged(QString);

// Emits the changed code word

private:

QString codeWord;

QLabel \*bullLabel;

QLabel \*cowLabel;

QLabel \*bullCount;

QLabel \*cowCount;

QLineEdit \*lineEdit;

QPushButton \*checkButton;

bool currentGuess;

};

#endif // GUESSENTRY\_H

#include <QMessageBox>

#include "guessentry.h"

GuessEntry::GuessEntry(QPixmap bull, QPixmap cow,

bool current, QWidget \*parent, Qt::WindowFlags f):QWidget(parent,f){

// Sets whether this is the current guess entry

currentGuess = current;

// Set the labels and images for the cow and bull labels

bullLabel = new QLabel(parent);

cowLabel = new QLabel(parent);

bullLabel->setPixmap(bull);

cowLabel->setPixmap(cow);

// Set the labels for the bull and cow counts

cowCount = new QLabel(parent);

bullCount = new QLabel(parent);

cowCount->setStyleSheet("color: blue; background-color: yellow; font: 18pt");

bullCount->setStyleSheet("color: blue; background-color: yellow; font: 18pt");

cowCount->setFixedSize(40,40);

cowCount->setAlignment(Qt::AlignCenter);

bullCount->setAlignment(Qt::AlignCenter);

bullCount->setFixedSize(40,40);

cowCount->setText("");

bullCount->setText("");

// Prepare the line edit and push button

lineEdit = new QLineEdit;

lineEdit->setStyleSheet("color:blue; background-color: yellow; font: 18pt");

lineEdit->setFixedWidth(100);

checkButton = new QPushButton(tr("Check"));

// By default set the button to hide and not enabled until

// the entry is current

lineEdit->setEnabled(false);

checkButton->hide();

checkButton->setEnabled(false);

// Set the guessEntry lineEdit to enable/disable the checkButton when it is filled/emptied

connect(lineEdit,SIGNAL(textChanged(const QString &)),this,SLOT(enableCheck(const QString &)));

// Set the checkButton to emit the guess when it is clicked

connect(checkButton,SIGNAL(clicked(bool)),this,SLOT(guessFinished(bool)));

// Create and set the layout

QGridLayout \*layout = new QGridLayout;

layout->addWidget(lineEdit,0,0);

layout->addWidget(bullLabel,0,1);

layout->addWidget(bullCount,0,2);

layout->addWidget(cowLabel,0,3);

layout->addWidget(cowCount,0,4);

layout->addWidget(checkButton,0,5);

layout->setMargin(1);

layout->setSizeConstraint(QLayout::SetFixedSize);

setLayout(layout);

}

// Sets the GuessEntry current variable as true, enables checkButton,

// and guessEntry lineEdit.

void GuessEntry::setCurrent(bool status){

if(status!=currentGuess){

currentGuess = status;

// Show the checkButton and enable the lineEdit

lineEdit->setEnabled(status);

checkButton->show();

emit current(currentGuess);

}

}

// Sets the code word for the game.

void GuessEntry::setCodeWord(QString code){

if(codeWord!=code){

codeWord = code;

emit codeWordChanged(code);

}

}

// Resets the current and displayed QLabels

void GuessEntry::reset(){

currentGuess = false;

bullCount->setText("");

cowCount->setText("");

lineEdit->clear();

lineEdit->setEnabled(false);

}

// Checks the lineEdit entry with the passed codeWord. It sets the

// cowCount and bullCount QLabels to the current bulls and cows

// in the code word and guess made by the user.

void GuessEntry::guessFinished(bool test){

// If neither the text nor the guess are empty check

if(!codeWord.isEmpty()&&currentGuess){

checkButton->hide();

lineEdit->setEnabled(false);

currentGuess = false;

QString guess = lineEdit->text().toUpper();

unsigned int cows = 0; // If letters are in the puzzle, but wrong spot

unsigned int bulls = 0; // If letters are in the puzzle and right spot

// Maps for the guess and puzzle

QSet<QChar> puzzleSet;

QSet<QChar> guessSet;

for (int i = 0; i < codeWord.length(); i++) {

if (guess[i] == codeWord[i])

bulls++;

// Fill both sets with the frequency value for each letter key

puzzleSet.insert(codeWord[i]);

guessSet.insert(guess[i]);

}

// Get the intersection between the puzzleSet and guessSet characters.

// The difference between the characters that are in the correct position

// and the size of the intersection will the number of characters that

// are in the code word, but wrong position.

QSet<QChar> intersection = puzzleSet.intersect(guessSet);

cows = intersection.size() - bulls;

bullCount->setText(QString::number(bulls));

cowCount->setText(QString::number(cows));

if(guess==codeWord){

// Send out if the code word is broken

emit solved(true);

}

else{

// The default value from QPushButton clicked() is false. We need to send opposite

// signal to show this guess entry is finished.

emit finished(!test);

}

// Send out the user entered guess

emit guessWord(guess,bulls, cows);

}

}

// Enables the check button only if the user has typed in

// valid text in the lineEdit control.

void GuessEntry::enableCheck(const QString &text){

// Enable the button if the text is not empty

checkButton->setEnabled(!text.isEmpty() && currentGuess);

}

// Changes the validation on the lineEdit based on the selected option

// from the difficulty QGroupButton class

void GuessEntry::setValidation(int test){

if(test>0){

// Set a regular expression to ensure guesses entered are only the

// required length from the radio buttons for difficulty

QRegExp guessExp(QString("[A-Za-z]{")+QString::number(test)+QString("}"));

lineEdit->setValidator(new QRegExpValidator(guessExp,this));

}

}

/\*

\* File: invalidargument.h

\* Author: Marlo Zeroth

\*

\* Created on November 9, 2015, 7:37 AM

\*/

#ifndef INVALIDARGUMENT\_H

#define INVALIDARGUMENT\_H

#include <iostream>

#include <cstdlib>

class InvalidArgument{

public:

InvalidArgument(){

std::cout <<"Argument is not valid." << std::endl;

exit (EXIT\_FAILURE);

};

};

#endif /\* INVALIDARGUMENT\_H \*/

#ifndef LIST\_H

#define LIST\_H

/\*

\* Author: Marlo Zeroth

\* List.h

\*

\* Implements a template doubly linked list data structure. Allows duplicates,

\* insertion at front, back, and any point in the list. The list is not ordered.

\* Created on: Nov 4, 2015

\*

\*/

#include <new>

#include "invalidargument.h"

#include "empty.h"

template<class T>

class List {

private:

// Internal struct class to hold data and point to other nodes

struct Node {

T data;

Node \*next;

Node \*previous;

};

/\*Utility functions\*/

void memError() const; // Handles memory allocation errors

void subError() const; // Handles subscripts out of range

// Returns the position of an entry in the list

typename List<T>::iterator position(const T& entry) const;

/\* DoublyList Member variables \*/

Node \*head; // Reference to first/head node

Node \*tail; // Reference to last/tail node

int size;

public:

// iterator class for traversing the list

class iterator {

public:

friend class List;

// Default constructor. Allows .

iterator();

// Copy constructor

//OListiterator(const iterator & obj);

// Destructor

//~OListiterator();

/\*Overloaded operators\*/

// Assignment operator

//iterator& operator = (const OListiterator& obj);

// Prefix increment operator

iterator& operator++();

// Postfix increment operator

iterator operator++(int);

// Dereference operator

T& operator\*();

// Returns the addres of the value referred to

T\* operator->();

// Comparison operator

bool operator==(const iterator &obj) const;

// Comparison operator

bool operator!=(const iterator &obj) const;

private:

iterator(Node \*pter);

Node \*position;

};

//Constructors and Destructor

List();

// Default constructor

List(const T &entry);

// Main constructor

List(const List<T> &obj);

// Copy constructor

~List();

// Destructor

// Overloaded Operators

List<T>& operator=(const List<T> &obj);

// Oveloaded = operator

/\* Mutators and Accessors \*/

void prepend(const T& entry);

// Adds an entry to the front of the list

void append(const T& entry);

// Adds an entry to the back of the list

void insertBefore(List<T>::iterator \*position, const T& entry);

// Inserts an element before the passed parameter

void insertAfter(List<T>::iterator \*position, const T& entry);

// Inserts an element after the passed parameter

T& first() throw (Empty);

// Returns the first element in the list

T& last() throw (Empty);

// Returns the last element in the list

iterator begin();

// Returns a pointer to the beginning of the list.

iterator begin() const;

// Returns a const iterator to the beginning of the list.

iterator end();

// Returns a null iterator to signify the end of the list.

iterator end() const;

// Returns a const null iterator to signify the end of the list.

bool contains(const T& entry);

// Returns whether the list contains the passed entry.

int length() const { return size;}

// Returns the size of the list

void remove(const T& entry);

// Removes the passed entry from the list if it exits. Throws

// an invalid argument if it does not exist.

void clear();

// Removes all items for the list

};

/\*DoublyList Constructor, Destructor, and Function Definitions \*/

// Default constructor

template<class T>

List<T>::List() {

tail = NULL;

head = NULL;

size = 0;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Main constructor. Adds an entry and increments the size by 1. \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

template<class T>

List<T>::List(const T &entry) {

// Create new node with entry parameter

Node \*link;

// Attempt to allocate memory

try {

link = new Node;

} catch (const std::bad\_alloc&) {

memError();

}

// Set its data member and pointers

link->data = entry;

link->next = NULL;

link->previous = NULL;

// Now point the head and tail to this new link in the list

head = tail = link;

size++;

}

// Copy constructor. This is order O(N).

template<class T>

List<T>::List(const List<T> &obj) {

std::cout << "copy constructor" << std::endl;

// Set the head and tail pointers to null.

tail = head = NULL;

// Set the size of the list

size = 0;

// Now set the cursor to the object's head and

// traverse the links in obj if there are any.

Node \*cursor; // For navigating through the obj list

for (cursor = obj.head; cursor != 0; cursor = cursor->next) {

append(cursor->data);

}

}

// Destructor

template<class T>

List<T>::~List() {

// Traverse the list deleting each link

while (head) {

// Set the cursor to the current head

Node \*cursor = head;

// Set the head pointer to the next link

head = head->next;

// Deallocate the current link

delete cursor;

}

tail = NULL;

}

// Overloaded operators

// Overload = operator declaration

template<class T>

List<T>& List<T>::operator=(const List<T> &obj) {

std::cout << "= operator " << std::endl;

// Test to make sure the passed object is not the same

// as calling object e.g. input = input

if (this != &obj) {

clear();

// Set the size of the list

size = 0;

// Now set the cursor to the object's head and

// traverse the links in obj if there are any.

Node \*cursor; // For navigating through the obj list

for (cursor = obj.head; cursor != 0; cursor = cursor->next) {

append(cursor->data);

}

}

// Just return a reference

return \*this;

}

/\*Mutators and accessors\*/

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// prepend Function. Adds a element at the beginning of the list \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

template<class T>

void List<T>::prepend(const T& input) {

// Create a new node for the input.

Node \*link;

// Allocate memory

try {

link = new Node;

} catch (const std::bad\_alloc&) {

memError();

}

// Set its data element.

link->data = input;

// If the list is empty, add the new link and point head and tail to it

if (!head) {

// Set the link's next and previous to null as this is the only link

link->next = NULL;

link->previous = NULL;

// Now point the head and tail to the new link

tail = head = link;

size++;

} else {

// Since it will be at the beginning, it should

// point to the current first link in the list.

link->next = head;

link->previous = NULL;

// Now set the previous pointer in the existing first link

// to the new link.

head->previous = link;

// Set the new head.

head = link;

size++;

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// append Function. Adds a element at the end of the list. \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

template<class T>

void List<T>::append(const T& input) {

// Create a new node for the input.

Node \*link;

// Allocate memory

try {

link = new Node;

} catch (const std::bad\_alloc&) {

memError();

}

// Set its data element.

link->data = input;

// If the list is empty, add the new link and point head and tail to it

if (!tail) {

// Set the link's next and previous to null as this is the only link

link->next = NULL;

link->previous = NULL;

// Now point the head and tail to the new link

tail = head = link;

size++;

} else {

// Since it will be at the end, it should

// point to the current last link in the list.

link->previous = tail;

link->next = NULL;

// Now set the next pointer in the existing last link

// to the new link.

tail->next = link;

// Set the new tail.

tail = link;

size++;

}

}

// Inserts an element before the passed parameter

template<class T>

void List<T>::insertBefore(List<T>::iterator \*location, const T& entry) {

Node \*cursor = const\_cast<Node\*> (location->position);

// Check to be sure this is a valid position

if (cursor == 0) {

throw InvalidArgument("invalid position");

}

Node \* const link;

try {

link = new Node;

} catch (const std::bad\_alloc&) {

memError();

}

link->data = entry;

// If the position is at the tail, change the tail

if (cursor == head) {

// Since it will be at the beginning, it should

// point to the current first link in the list.

link->next = head;

link->previous = NULL;

// Now set the previous pointer in the existing first link

// to the new link.

head->previous = link;

// Set the new head.

head = link;

size++;

} else {

Node \*prevPtr = head;

while (prevPtr != 0 && prevPtr->next != cursor) {

prevPtr = prevPtr->next;

}

if (prevPtr == 0) {

throw InvalidArgument("invalid position");

}

prevPtr->next = link;

}

}

// Inserts an element after the passed parameter

template<class T>

void List<T>::insertAfter(List<T>::iterator \*location, const T& entry) {

// Check to be sure this is a valid position

if (location == 0) {

throw InvalidArgument("invalid position");

}

Node \*link;

try {

link = new Node;

} catch (const std::bad\_alloc&) {

memError();

}

link->data = entry;

location->position->next = link;

link->previous = location.position;

// If the position is at the tail, change the tail

if (location->position == tail)

tail = link;

}

// Returns the first element in the list

template<class T>

T& List<T>::first() throw (Empty) {

if (head) {

return head->data;

} else {

throw Empty();

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// last function. Returns the last element in the list \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

template<class T>

T& List<T>::last() throw (Empty) {

if (tail) {

return tail->data;

} else {

throw Empty();

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// begin function. Returns an iterator to the beginning of the list \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

template<class T>

typename List<T>::iterator List<T>::begin() {

// Create an iterator and set its

// position to the head.

return iterator(head);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// begin function. Returns an iterator to the beginning of the list \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

template<class T>

typename List<T>::iterator List<T>::begin() const {

// Create an iterator and set its

// position to the head.

return iterator(head);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// end function. Returns an iterator to the end of the list \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

template<class T>

typename List<T>::iterator List<T>::end() {

// Create an iterator and set its position to the

// tail.

return iterator(NULL);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// end function. Returns an iterator to the end of the list \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

template<class T>

typename List<T>::iterator List<T>::end() const {

// Create an iterator and set its position to the

// tail.

return iterator(NULL);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// contains function. Returns true if entry is in the list, false \*

// otherwise \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

template<class T>

bool List<T>::contains(const T& entry) {

Node \*cursor = head;

bool found = false;

// Go through the list until we reach the end

// or find the entry

while (!found && cursor != 0) {

if (cursor->data == entry)

found = true;

cursor = cursor->next;

}

return found;

}

template<class T>

void List<T>::remove(const T& entry){

// For traversing the list

Node \*cursor = head;

// For changing the link before the entry to delete

Node \*prevPtr = NULL;

bool found = false;

// Traverse the list until we find the node or

// we reach the end

while (cursor->data != entry && cursor != NULL) {

if (cursor->data == entry)

found = true;

// Set the previous link to the current link

prevPtr = cursor;

// Go to the next link

cursor = cursor->next;

}

// If it is not in the list throw an exception

if (cursor == 0)

throw InvalidArgument("Entry not in the list");

// If we found it delete it

if (cursor == head) {

// If it is at the front of the list, there is no previous

// link. Just set the head to the next link

head = cursor->next;

found = true;

} else {

// Link the previous node and the node after the

// one to delete

prevPtr->next = cursor->next;

found = true;

}

// Deallocate it

delete cursor;

--size;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// clear function. Deallocates each link in the list and sets the size to 0 \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

template<class T>

void List<T>::clear() {

Node \*cursor;

// Clear out the data in the existing list

while (head) {

// Set the cursor to the current head

cursor = head;

// Set the head pointer to the next link

head = head->next;

// Deallocate the current link

delete cursor;

}

tail = NULL;

size = 0;

}

/\*iterator definitions\*/

// Default constructor

template<class T>

List<T>::iterator::iterator() {

}

// Main constructor

template<class T>

List<T>::iterator::iterator(Node \*ptr) {

position = ptr;

}

/\*Overloaded operators\*/

// != operator

template<class T>

bool List<T>::iterator::operator!=(const iterator &obj) const {

return position != obj.position;

}

// == operator

template<class T>

bool List<T>::iterator::operator==(const iterator &obj) const {

return position == obj.position;

}

// Dereferences the pointer and returns value pointed by position

template<class T>

T& List<T>::iterator::operator\*() {

return position->data;

}

template<class T>

T\* List<T>::iterator::operator->() {

return (&\*(List<T>::iterator) \* this);

}

// Prefix increments the iterator to the next element

template<class T>

typename List<T>::iterator& List<T>::iterator::operator++() {

position = position->next;

return \*this;

}

// Postfix increments the iterator to the next element

template<class T>

typename List<T>::iterator List<T>::iterator::operator++(int) {

iterator temp = \*this;

++(\*this);

return temp;

}

/\*Utility Functions\*/

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// memError function. Displays an error message and \*

// terminates the program when memory allocation fails. \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

template<class T>

void List<T>::memError() const {

std::cout << "ERROR:Cannot allocate memory.\n";

exit(EXIT\_FAILURE);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// subError function. Displays an error message and \*

// terminates the program when a subscript is out of range. \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

template<class T>

void List<T>::subError() const {

std::cout << "ERROR: Subscript out of range.\n";

exit(EXIT\_FAILURE);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Returns the position of an entry in the list. \*

// If the entry is not in the list, it returns null. \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

template<class T>

typename List<T>::iterator List<T>::position(const T& entry) const {

iterator cursor(head);

iterator finish(NULL);

while (\*cursor != entry && cursor != finish) {

cursor++;

}

return cursor;

}

#endif /\* LIST\_H \*/

#ifndef LOGINDIALOG\_H

#define LOGINDIALOG\_H

#include <QDialog>

#include <QRegExp> // For testing regular expressions

#include <QRegExpValidator>

#include <QLabel>

#include <QLineEdit>

#include <QCheckBox>

#include <QDialogButtonBox>

#include <QGridLayout>

class LoginDialog : public QDialog

{

Q\_OBJECT

public:

explicit LoginDialog(QWidget \*parent = 0);

//~LoginDialog();

private slots:

//void checkUserPassword();

signals:

void acceptLogin(QString& username, QString& password, int& indexNumber);

public slots:

void slotAcceptLogin(bool);

private:

QLabel \*passwordLabel;

QLabel \*userLabel;

QLineEdit \*userNameLine;

QLineEdit \*passwordLine;

QCheckBox \*rememberCheckBox;

QDialogButtonBox \*buttonBox;

};

#endif // LOGINDIALOG\_H

#include "logindialog.h"

#include <QMessageBox>

#include <QPushButton>

#include <QGridLayout>

LoginDialog::LoginDialog(QWidget \*parent) :

QDialog(parent)

{

// Regular expressions to check valid user name and input

QRegExp userExp("^[a-z0-9\_-@.]{3,15}$");

QRegExp passwordExp("((?=.[0-9])(?=.\*[a-z])(?=.\*[A-Z])(?=.\*[@#$%]).{8,20})");

// Initialize the username and password line edits

userNameLine = new QLineEdit(this);

passwordLine = new QLineEdit(this);

// Set up up the validators for the user and password lines

userNameLine->setValidator(new QRegExpValidator(userExp,this));

passwordLine->setValidator(new QRegExpValidator(passwordExp,this));

// Set the password line so that it does not show characters

passwordLine->setEchoMode(QLineEdit::Password);

// Initialize the labels and set their buddys

userLabel = new QLabel(this);

passwordLabel = new QLabel(this);

userLabel->setText((tr("Username")));

passwordLabel->setText((tr("Password")));

userLabel->setBuddy(userNameLine);

passwordLabel->setBuddy(passwordLine);

// Initialize and set up the buttons

buttonBox = new QDialogButtonBox(this);

buttonBox->addButton(QDialogButtonBox::Ok);

buttonBox->addButton(QDialogButtonBox::Cancel);

buttonBox->button(QDialogButtonBox::Ok)->setText(tr("Login"));

buttonBox->button(QDialogButtonBox::Cancel)->setText(tr("Cancel"));

// Connect the buttons slots and signals

connect(buttonBox->button(QDialogButtonBox::Cancel),SIGNAL(clicked()),this,SLOT(close()));

connect(buttonBox->button(QDialogButtonBox::Ok),SIGNAL(clicked()),this,SLOT(slotAcceptLogin()));

// Now create and set the layout

QGridLayout \*formLayout = new QGridLayout(this);

formLayout->addWidget(userLabel,0,0);

formLayout->addWidget(userNameLine,0,1);

formLayout->addWidget(passwordLabel,1,0);

formLayout->addWidget(passwordLine,1,1);

formLayout->addWidget(buttonBox,2,0,1,2);

setLayout(formLayout);

// Set the window title and whether it is modal>

setWindowTitle(tr("User Login"));

setModal(true);

}

void LoginDialog::slotAcceptLogin(bool){

QMessageBox msgBx;

msgBx.setText("Pressed Ok");

msgBx.*exec*();

}

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <QAction>

#include <QMenu>

#include "playspace.h"

#include "logindialog.h"

#include "stack.h"

class MainWindow:public QMainWindow

{

Q\_OBJECT

public:

MainWindow();

private slots:

void newGame();

void about();

void login(); // Opens the login dialog

private:

void createActions();

void createMenus();

void createContextMenu();

void createToolBars();

void createStatusBar();

void updateStatusBar();

const unsigned int TRIES;

const unsigned int DEFAULTDIFFICULTY;

QLabel \*loginLabel;

QLabel \*currentMoveLabel;

LoginDialog \*loginDialog;

PlaySpace \*playSpace;

Stack<QString> guesses;

QMenu \*fileMenu;

QMenu \*helpMenu;

QToolBar \*fileToolBar;

QAction \*separatorAction;

QAction \*newAction;

QAction \*saveAction;

QAction \*saveAsAction;

QAction \*loginAction;

QAction \*exitAction;

QAction \*aboutAction;

QAction \*aboutQtAction;

};

#endif // MAINWINDOW\_H

#include <QMessageBox>

#include <QMenuBar>

#include <QToolBar>

#include <QStatusBar>

#include <QApplication>

#include "mainwindow.h"

MainWindow::MainWindow(): TRIES(10),DEFAULTDIFFICULTY(3)

{

QString image1(":/images/Bull-icon.png");

QString image2(":/images/cow-icon.png");

QPixmap bull(image1);

QPixmap cow(image2);

playSpace = new PlaySpace(bull, cow, TRIES);

loginDialog = new LoginDialog;

createActions();

createMenus();

createContextMenu();

createToolBars();

createStatusBar();

setCentralWidget(playSpace);

setWindowIcon(QIcon(":/images/bull-icon.png"));

}

void MainWindow::login(){

loginDialog->show();

}

void MainWindow::newGame(){

MainWindow \*window = new MainWindow;

window->show();

}

void MainWindow::about()

{

QMessageBox::about(this, tr("About Bulls and Cows"),

tr("<h2>Bulls and Cows 1.1</h2>"

"<p>Copyright &copy; 2015 Marlo Zeroth."

"<p>Bulls and Cows is a small application that "

"demonstrates QNetwork, QMySQL, QAction, QMainWindow, QMenuBar, "

"QStatusBar, QTableWidget, QToolBar, and many other "

"Qt classes."));

}

void MainWindow::createActions(){

newAction = new QAction(tr("&New"), this);

newAction->setIcon(QIcon(":/images/new.png"));

newAction->setShortcut(QKeySequence::New);

newAction->setStatusTip(tr("Create a new bulls and cows game file"));

connect(newAction, SIGNAL(triggered()), this, SLOT(newGame()));

saveAction = new QAction(tr("&Save"), this);

saveAction->setIcon(QIcon(":/images/save.png"));

saveAction->setShortcut(QKeySequence::Save);

saveAction->setStatusTip(tr("Save the game to disk"));

connect(saveAction, SIGNAL(triggered()), this, SLOT(save()));

saveAsAction = new QAction(tr("Save &As..."), this);

saveAsAction->setStatusTip(tr("Save the bulls and cows under a new "

"name"));

connect(saveAsAction, SIGNAL(triggered()), this, SLOT(saveAs()));

loginAction = new QAction(tr("L&ogin"),this);

loginAction->setIcon(QIcon(":/images/login-icon.png"));

loginAction->setShortcut(tr("Ctrl+L"));

loginAction->setStatusTip(tr("Login to the remote server"));

connect(loginAction,SIGNAL(triggered()),this,SLOT(login()));

exitAction = new QAction(tr("E&xit"), this);

exitAction->setShortcut(tr("Ctrl+Q"));

exitAction->setStatusTip(tr("Exit the application"));

connect(exitAction, SIGNAL(triggered()), this, SLOT(close()));

aboutAction = new QAction(tr("&About"), this);

aboutAction->setStatusTip(tr("Show the application's About box"));

connect(aboutAction, SIGNAL(triggered()), this, SLOT(about()));

aboutQtAction = new QAction(tr("About &Qt"), this);

aboutQtAction->setStatusTip(tr("Show the Qt library's About box"));

connect(aboutQtAction, SIGNAL(triggered()), qApp, SLOT(aboutQt()));

}

void MainWindow::createMenus(){

fileMenu = menuBar()->addMenu(tr("&File"));

fileMenu->addAction(newAction);

fileMenu->addAction(saveAction);

fileMenu->addAction(saveAsAction);

fileMenu->addAction(loginAction);

separatorAction = fileMenu->addSeparator();

fileMenu->addSeparator();

fileMenu->addAction(exitAction);

menuBar()->addSeparator();

helpMenu = menuBar()->addMenu(tr("&Help"));

helpMenu->addAction(aboutAction);

helpMenu->addAction(aboutQtAction);

}

void MainWindow::createContextMenu(){

}

void MainWindow::createToolBars(){

fileToolBar = addToolBar(tr("&File"));

fileToolBar->addAction(newAction);

fileToolBar->addAction(saveAction);

fileToolBar->addAction(loginAction);

}

void MainWindow::createStatusBar(){

loginLabel = new QLabel(" Local Player - Not Logged in ");

loginLabel->setAlignment(Qt::AlignHCenter);

loginLabel->setMinimumSize(loginLabel->*sizeHint*());

currentMoveLabel = new QLabel;

currentMoveLabel->setIndent(3);

statusBar()->addWidget(loginLabel);

statusBar()->addWidget(currentMoveLabel, 1);

updateStatusBar();

}

void MainWindow::updateStatusBar(){

}

#include <QApplication>

#include "mainwindow.h"

int main(int argc, char \*argv[])

{

QApplication a(argc,argv);

MainWindow w;

w.show();

return a.exec();

}

#ifndef PLAYSPACE\_H

#define PLAYSPACE\_H

#include <QWidget>

#include <QString>

#include <QList>

#include <QStack>

#include <QVector>

#include <QPixmap>

#include <QComboBox>

#include <QGroupBox>

#include <QButtonGroup>

#include <QRadioButton>

#include <QStringList>

#include <QVBoxLayout>

#include <QHBoxLayout>

// User libraries

#include "guessentry.h"

#include "list.h"

class PlaySpace:public QWidget{

Q\_OBJECT

public:

// Main constructor

PlaySpace(QPixmap bull, QPixmap cow, unsigned int size, QWidget \*parent=0, Qt::WindowFlags f=0);

public slots:

// Slot to pass the code word to GuessEntry object

//void startPlay(QString code, int difficulty);

private slots:

void setRole(int role);

// Hides and unhides the code lineEdit if user is breaker or creator

void setValidation(int test);

// Sets the validation for the lineEdit for code word.

void newGame(bool test);

// Starts a new game

signals:

//void myHandler(QString codeWord);

void codeChanged(QString code);

void gameFinished(bool );

private:

QPixmap cow; // To hold images for cows

QPixmap bull; // To hold images for bulls

QString codeWord; // Holds the current code to break

unsigned int currentGuess; // Holds the current guess made

unsigned int totalGuesses; // Holds how many guess entries to generate

bool finished; // If the current game is complete

List<GuessEntry \*> guessInterface; // Holds the guessing interface.

// For the new game entry

QGroupBox \*role;

QLabel \*codeLabel;

QLineEdit \*lineEdit;

QButtonGroup \*difficulty;

QGroupBox \*difficultyRadioBox;

QComboBox \*playerRole;

QPushButton \*newGameButton;

QPushButton \*quitButton;

};

#endif // PLAYSPACE\_H

#ifndef PLAYSPACE\_CPP

#define PLAYSPACE\_CPP

#include <QMessageBox>

#include "playspace.h"

PlaySpace::PlaySpace(QPixmap bull, QPixmap cow, unsigned int size, QWidget \*parent, Qt::WindowFlags f):QWidget(parent,f){

totalGuesses = size;

currentGuess = 0;

// Set the difficulty

difficulty = new QButtonGroup;

difficultyRadioBox = new QGroupBox(tr("Difficulty:"));

QRadioButton \*easy = new QRadioButton(tr("Easy: "));

QRadioButton \*medium = new QRadioButton(tr("Medium:"));

QRadioButton \*hard = new QRadioButton(tr("Hard:"));

QRadioButton \*harder = new QRadioButton(tr("Harder:"));

medium->setChecked(true);

// Add the buttons to the difficulty button group

difficulty->addButton(easy);

difficulty->addButton(medium);

difficulty->addButton(hard);

difficulty->addButton(harder);

// Set the values for the difficulty

difficulty->setId(easy,3);

difficulty->setId(medium,4);

difficulty->setId(hard,5);

difficulty->setId(harder,6);

// Create a layout for the difficulty

QVBoxLayout \*diffLayout = new QVBoxLayout;

diffLayout->addWidget(easy);

diffLayout->addWidget(medium);

diffLayout->addWidget(hard);

diffLayout->addWidget(harder);

difficultyRadioBox->setLayout(diffLayout);

role = new QGroupBox(tr("Player Role"));

// Prepare the playerRole combo box

playerRole = new QComboBox(parent);

QStringList options;

options<< "Breaker" <<"Creator";

playerRole->addItems(options);

// Set the code label and line edit for the code word

codeLabel = new QLabel(parent);

codeLabel->setText("Enter Code Word: ");

lineEdit = new QLineEdit;

lineEdit->setStyleSheet("color:blue; background-color: yellow; font: 18pt");

lineEdit->setFixedWidth(100);

codeLabel->setBuddy(lineEdit);

QVBoxLayout \*roleLayout = new QVBoxLayout;

roleLayout->addWidget(playerRole);

roleLayout->addWidget(codeLabel);

roleLayout->addWidget(lineEdit);

// Hide the codeLabel and lineEdit.

codeLabel->*setVisible*(false);

lineEdit->*setVisible*(false);

// Set it so characters are hidden like in password entry

lineEdit->setEchoMode(QLineEdit::Password);

role->setLayout(roleLayout);

newGameButton = new QPushButton("New Game",parent);

connect(playerRole,SIGNAL(currentIndexChanged(int)),this,SLOT(setRole(int)));

connect(newGameButton,SIGNAL(clicked(bool)),this,SLOT(newGame(bool)));

connect(difficulty,SIGNAL(buttonPressed(int)),this,SLOT(setValidation(int)));

QHBoxLayout \*gameLayout = new QHBoxLayout;

gameLayout->addWidget(difficultyRadioBox);

gameLayout->addWidget(role);

gameLayout->addWidget(newGameButton);

QVBoxLayout \*bottomLayout = new QVBoxLayout;

bottomLayout->addLayout(diffLayout);

bottomLayout->addLayout(roleLayout);

bottomLayout->addLayout(gameLayout);

QVBoxLayout \*rightLayout = new QVBoxLayout;

// Create GuessEntry objects and connect the bullsCows signal to their setBullsCows slots

for(unsigned int i = 0; i<size;i++){

GuessEntry \*entry = new GuessEntry(bull, cow, false,parent,f);

rightLayout->addWidget(entry);

connect(this,SIGNAL(codeChanged(QString)),entry,SLOT(setCodeWord(QString)));

connect(difficulty,SIGNAL(buttonPressed(int)),entry,SLOT(setValidation(int)));

guessInterface.prepend(entry);

}

// Iterators for current and its next GuessEntry object in the list

List<GuessEntry\*>::iterator next = guessInterface.begin();

List<GuessEntry\*>::iterator curr = next++;

List<GuessEntry\*>::iterator finish = guessInterface.end();

// Go through each guessInterface object in the list and connect it to its adjacent

// object

while(next!=finish){

connect(\*curr,SIGNAL(finished(bool)),\*next,SLOT(setCurrent(bool)));

curr++;

next++;

}

rightLayout->addLayout(bottomLayout);

// Add the game start button and difficulty comboboxes

// Set the first guess entry as the current entry

setLayout(rightLayout);

//guessInterface.first()->setCurrent(true);

}

// Set the label and lineEdit for getting the code word

// visible or invisible

void PlaySpace::setRole(int role){

if(role==1){

codeLabel->*setVisible*(true);

lineEdit->*setVisible*(true);

}else{

codeLabel->*setVisible*(false);

lineEdit->*setVisible*(false);

}

}

// Changes the validation on the lineEdit based on the selected option

// from the difficulty QGroupButton class

void PlaySpace::setValidation(int test){

if(test>0){

// Set a regular expression to ensure guesses entered are only the

// required length from the radio buttons for difficulty

QRegExp guessExp(QString("[A-Za-z]{")+QString::number(test)+QString("}"));

lineEdit->setValidator(new QRegExpValidator(guessExp,this));

}

}

// Starts a new game with the entered code word and difficulty

// signal

void PlaySpace::newGame(bool test){

if(!test){

QMessageBox msgBx;

msgBx.setText("Test is true");

msgBx.*exec*();

}

if(!test){

// Set the code word for the game

QString code = lineEdit->text().toUpper();

emit codeChanged(code);

// Reset the current status of the individual GuessEntry objects

// Iterators for current and its next GuessEntry object in the list

List<GuessEntry\*>::iterator curr = guessInterface.begin();

List<GuessEntry\*>::iterator finish = guessInterface.end();

// Go through each guessInterface object in the list and reset current to false

while(curr!=finish){

(\*curr)->reset();

curr++;

}

guessInterface.first()->setCurrent(true);

}

}

/\*

void PlaySpace::resetPlay(){

}

\*/

#endif // PLAYSPACE\_CPP