Mark Gallagher | mag0038@auburn.edu

COMP2710: Project 1 – Phase 2

April 4, 2015

Note: Some things are subject to change. Example: data types for player attributes may change from int to short or u\_int or u\_short. This will probably occur during phase 3 if a change is needed.

System-level Tests:

Test Cases:

|  |  |  |  |
| --- | --- | --- | --- |
| **What** | **Input** | **Expected Output** | **Actual Output** |
| Entering a valid username string | “GLaDOS” | None. Redirects you to system menu. |  |
| Entering an invalid username string | “” (empty string) | “Please enter a valid username (longer than 1 character). Try again” |  |
| Entering a valid system-menu option | 2 | None. Redirects to appropriate option |  |
| Entering an negative number for menu option | -1 | “Option does not exist. Remember to enter 0 or positive numbers only. Try again” |  |
| Entering an out of range menu option | 55 | “Option does not exist. Try again” |  |
| Entering a letter or string when an integer is needed | GG | “Please enter numbers only, not letters or symbols. Try again” |  |
| Entering a letter when an integer is expected (some puzzles) | A | “Please enter numbers only, not letters or symbols. Try again” |  |
| Entering a valid puzzle answer (correct) | Donuts | “You are correct!” |  |
| Entering a valid puzzle answer (wrong) | Bagels | “Your answer is wrong! :(” |  |
| User enters a valid filename for load/save scores | score1.txt | “Program will load and save scores from file ‘score1.txt’” |  |
| User enters nothing for filename | “” (empty string) | “ERROR: No name entered. Program will continue with default name ‘scoreboard.txt’” |  |
| File I/O error |  | “ERROR: Program could not load (save) from file <filename>.” |  |
| A player attribute reaches 0 (game end) |  | “Game Over” |  |
| Player attribute goes negative |  | “Game Over” (program should treat it as 0 |  |

Algorithm Design:

Prompts user to login, saves user’s name

@return user’s name through pass-by-reference

void login\_process(string& userName)

{

print welcome message;

while userName not valid (as in, length < 1)

{

prompt user to enter name;

save entered name to userName;

}

}

Initializes character to appropriate values: Provided name, random attributes (within range), preset steps from goal

@return an initialized player structure through pass-by-reference

void init\_character(string userName, player& userPlayer)

{

set userPlayer.name to userName;

set userPlayer.money to rand() int in valid range (TBD);

set userPlayer.intel to rand() int in valid range;

set userPlayer.time to rand() int in valid range (different from other attribute ranges);

set userPlayer.steps to preset value;

set userPlayer.score to 0;

}

Initializes puzzle list to preset values << not from a file >>

@return initialized puzzle list with preset values through pass-by-reference

void create\_puzzle\_list(puzzleHead& p\_head)

{

set p\_head.numOfPuzzles to however many puzzles there are (preset, a constant);

set p\_head.first to new puzzleNode;

set first puzzleNode’s properties according to presets;

repeat by adding more nodes to previous node’s “next” attribute until all puzzles have been created (ALL HARD CODED. Loop could be used, but may make code really hard to read/follow (example: unit test code for this function));

set last puzzleNode.next = NULL;

}

Initialize puzzle list according to a user-provided input file

@return initialized puzzle list from file through pass-by-reference

@return non-zero integer if error occurs

int create\_puzzle\_list(puzzleHead& p\_head, const char filename[])

{

have an iterator to move through list, set it to p\_head.first;

open puzzle file;

return 1 if error in opening file;

if p\_head.first == NULL, need to obtain first node from file, so:

read data from file;

create new node from file data;

set p\_head.first = new node;

while (not at end of file)

{

read puzzle data from file;

new puzzleNode = create a puzzle node;

insert puzzleNode to list;

increment p\_head.numOfPuzzles;

update iterator to point to node just added;

}

close file;

return 0;

}

Initializes encounters to preset array

@return encounter array through pass-by-reference

void init\_encounters(encounterPtr& encArray)

{

allocate memory to pointer into a dynamic array;

assign each element of array to a new encounter structure << HARD CODED since each encounter structure is unique >>;

}

Displays scores to user

void view\_scores(scoreHead sBoard)

{

display how many scores in scoreboard;

while (not at end of list)

{

display score in a ‘formatted’ output;

}

display message “no more scores to show”;

}

Loads scores from a file (in this case, filename not provided, use default)

@return initialized scoreboard list

@return non-zero integer if error occurs

int load\_scores(scoreHead& sBoard)

{

// since operation same as loading from a file with user-given name, can just call function using the default name

return load\_scores(sBoard, “DEFAULT FILE NAME HERE”);

}

Loads scores from a file with a given filename – IF USER PROVIDED A FILENAME, SHOULD BE SAVED FOR USE WITH CORRESPONDING “save\_scores” METHOD

@return initialized scoreboard through pass-by-reference

@return non-zero integer if error occurs

int load\_scores(scoreHead& sBoard, const char filename[])

{

open a score file;

return 1 if error in opening file;

while (not at end of file)

{

read a score from file;

new\_score\_node = create a score node from data in file;

insert new\_score\_node to end of list;

}

close file;

return 0;

}

Saves scores to default file

@return non-zero integer if error occurs

int save\_scores(scoreHead sBoard)

{

// Since operation same as loading from a file with user-given name, just call function with default

return save\_scores(sBoard, “DEFAULT FILE NAME HERE”);

}

Saves scores to a file with given filename

@return non-zero integer if error occurs

int save\_scores(scoreHead sBoard, const char filename[])

{

create an iterator pointer to traverse sBoard list;

open a score file;

return 1 if error occurs;

write data from sBoard header structure (how many in list, etc);

while (not at end of scoreboard list)

{

write current score node to file;

update iterator to next score node;

}

close file;

return 0;

}

Adds a new score to the scoreboard list, in sorted order (first node is HIGHEST)

@return non-zero integer if operation not successful (e.g., user’s score didn’t make the cut, or invalid score (0))

int add\_new\_score(scoreHead& sBoard, int userScore, string userName)

{

if score <= 0

{

return 1; // cannot add this to list

}

// Follow Dr. Qin’s example for adding data in singly linked list

create two pointers to iterate through list, pre and curr;

create new\_ptr and assign a new node to it;

initialize new node to appropriate values as given;

set new\_ptr->next = NULL;

if (root == NULL) // list empty, add to start

{

root = new\_ptr;

}

else // insert node into sorted list

{

pre\_ptr = NULL; cur\_ptr = root;

while ((cur\_ptr != NULL) && (curr\_ptr->score > info))

{

pre\_ptr = cur\_ptr;

cur\_ptr = cur\_ptr->next;

}

if (pre\_ptr == NULL) // insert to head of list

{

new\_ptr->next = root;

root = new\_ptr;

}

else // non-head node

{

new\_ptr->next = cur\_ptr;

pre\_ptr->next = new\_ptr;

}

}

return 0;

}

Calculates the user’s score and sets it in their player profile

@return player profile now has score set

void calculate\_score(player& userPlayer)

{

int calcScore = userPlayer.money \* userPlayer.time \* userPlayer.intel;

if (calcScore > 0)

{

userPlayer.score = calcScore;

}

else

{

userPlayer.score = 0;

}

}

Prompts user to enter a number and ensure that the number entered is 1) not alpha character and 2) nonnegative.

@return user-entered positive integer

int valid\_int\_input()

{

// Use Dr. Qin’s presentation on obtaining string and non-string data

int result = -1;

while (result < 0) // while result is NOT valid

{

prompt user to enter number;

if (!(cin >> result)) // cin is NOT successful (string entered)

{

remind user to enter numbers only;

cin.clear();

cin.ignore(INT\_MAX, ‘\n’); // large num instead of INT\_MAX works too

}

if (result < 0)

{

remind user to enter 0 or positive numbers only;

}

}

return result;

}

Displays character information to user

void view\_character(player userPlayer)

{

print all of userPlayer information (except score) in formatted output;

}

Makes player read a paper, modify attributes accordingly

@return modified player

void read\_paper(player& userPlayer)

{

print story info about the paper;

set userPlayer.intel += rand() in valid range;

set userPlayer.time -= rand() in valid range;

print what player attributes changed by how much;

}

Makes player search for change, modify attributes accordingly

@return modified player

void search\_change(player& userPlayer)

{

print story info about looking for change;

set userPlayer.money += rand() in valid range;

set userPlayer.time -= rand() in valid range;

print what player attributes changed by how much;

}

Easter egg event, ends game…

void easter\_egg\_event(player& userPlayer)

{

print that player found secret event;

print that they got a huge raise;

print that however, they were abducted and taken into an alternate reality;

set userPlayer.money += a large number;

set all other attributes to 0;

print character info one last time;

terminate program;

}

Moves player forward (at least, most events will) by 1. Will affect different things depending on encounter generated

@return updated player attributes

void move\_forward(player& userPlayer, encounterPtr encArray, puzzleHead p\_head)

{

encounter\_t encID = generate\_encounted\_ID(); // not sure, can you use an int to an enumerated type???

print user moves forward and…;

switch (encID)

{

case NOTHING:

print nothing happened;

update according to what’s in encArray; // MIGHT refactor this step into a function, time depending

print what changed;

break;

case PUZZLE:

print ITS PUZZLE TIME!;

call function: ask\_puzzle(userPlayer, p\_head);

break;

case PROFESSOR:

print professor encounter;

update according to what’s in encArray;

print what changed;

break;

case STUDENT:

print student encounter;

update according to what’s in encArray;

print what changed;

break;

case GRUNT:

print work time;

update according to what’s in encArray;

print what changed;

break;

case GRADE:

print grading papers;

update according to what’s in encArray;

print what changed;

break;

default:

break; // this shouldn’t happen

}

}

Asks the user a puzzle question and then updates player's attributes

@return modified player attributes

void ask\_puzzle(player& userPlayer, puzzleHead p\_head)

{

create pointer to traverse puzzle list;

u\_short puzzleID = rand() in valid range;

for (i = 0; i < puzzleID; i++)

{

traverse puzzle list;

}

print puzzle question;

prompt user to enter answer;

bool isCorrect = compare userAnswer with puzzleAnswer;

if (isCorrect)

{

print they got it right message;

}

else

{

print they got it wrong (sadface);

}

modify attributes according to what’s in action array;

print what changed;

}

Displays the system menu (after user logs in)

void display\_sys\_menu()

{

print menu with integers representing valid user options;

}

Displays the game menu (when a game is started)

void display\_game\_menu()

{

print menu with integers representing valid user options;

}