**Data Structures and Algorithms**

**Database of Games**

**An End-Semester Project by**

**Hamaz Hamza 365670**

**Omer Farooq 377187**

**Syed Hamza Ali Shah 396358**

**Introduction**

In our project, we have made a program that stores data about games across many different years, genres, developers, platforms, etc. Our dataset includes 16720 rows and 15 columns. The data we have chosen also lists each game’s sales across the world, scores and user amounts by both critics and gamers. We have made 19 functions that allows the user to retrieve many different types of data from out dataset, and these will be described later. We have used graphs, AVL trees, doubly linked lists, arrays, and hash tables to store our data, these will be also be described later on.

**Data Structures**

|  |  |  |
| --- | --- | --- |
| **#** | **Data Structure** | **Purpose** |
| 1 | Graph | Storing the names and lists of related games for each game within the program. We have not used a hash table because hundreds of new games are added each year, so a hash table would not be efficient in this case. Our graph has O(log n) searching. |
| 2 | Hash Table | Storing all the games of a certain platform within a specific platform node within the hash table. We have used a hash table to allow O(1) searching and retrieval. |
| 3 | Hash Table | Storing all the games of a certain genre within a specific genre node within the hash table. We have used a hash table to allow O(1) searching and retrieval. |
| 4 | Hash Table | Storing all the games of a certain publisher within a specific publisher node within the hash table. We have used a hash table to allow O(1) searching and retrieval. |
| 5 | Hash Table | Storing all the games of a certain developer within a specific developer node within the hash table. We have used a hash table to allow O(1) searching and retrieval. |
| 6 | Array | Storing all the games of a certain year within a specific year array node within the array. We have used an array because we can easily map the year to the node’s index within the array by subtracting 1980 from the year. |
| 7 | Array | Storing all the games of a certain rating within a rating array node within the array. We have used an array because we can easily map the rating to the node’s index within the array, by using 8 if statements, as there are only 8 possible ratings. |
| 8 | AVL Tree | To store the global sales data. Each node will have a linked list that contains all games that have the same global sales data. We have used an AVL tree to allow faster searching times. |
| 9 | Doubly Linked Lists | A doubly linked list has been included in the above data structures. We have also used multiple lists to store games sorted by numeric data e.g. user score, user count, critic score, critic count etc. We have used a doubly linked list as it allows us to get data both in ascending and descending order |

**Functions:**

**Important Note: We have made 19 different functions. Each function has a lot of helping functions so please refer to the source code for complete understanding.**

We are going to show each function with the help of the interface we designed. Here is the code for the interface:

while (true) {

cout << endl;

cout << "Press the associated number to select one of the following option:" << endl << "1. Get the info for a game" << endl << "2. Get games for a certain parameter" << endl;

cout << "3. Get recommendations for similar Games" << endl << "4. Get a list of the best Games" << endl << "Enter 0 to exit" << endl;

cin >> mode;

if (mode == 0)break;

cout << endl;

if (mode == 1) {

while (true) {

cout << endl;

cout << "Enter the name of the game or enter y to go back" << endl;

cin.ignore();

getline(cin, val);

if (val == "y")break;

cout << endl;

gamesgraph->info(val);

cout << endl;

cout << "Press 1 to go back or 2 to find info for another game" << endl;

cin >> amount;

if (amount == 1)break;

}

}

else if (mode == 2) {

while (true) {

cout << endl;

cout << "Press the associated number to select one of the following option:" << endl << "1. Get the best Games for a Platform" << endl;

cout << "2. Get the best games for a year" << endl << "3. Get the best Games for a certain Genre" << endl << "4. Get the best Games for a certain Age Rating" << endl;

cout << "5. Get the best game from a certain Developer" << endl << "6. Get the best Games for a certain publisher" << endl << "Enter 0 to go back" << endl;

cin >> mode2;

if (mode2 == 0)break;

cout << endl;

if (mode2 == 1) {

while (true) {

cout << endl;

cout << "Enter the name of the Platform or Enter y to go back" << endl;

cin.ignore();

getline(cin, val);

if (val == "y")break;

cout << endl;

cout << "Enter the number of games you want" << endl;

cin >> amount;

cout << endl;

cout << "Enter 1 to get Best games by sales and 2 to get Best Games by User Score" << endl;

cin >> mode2;

cout << endl;

if (mode == 1)platformTable->platformSales(val, amount);

else if (mode == 2) platformTable->platformScore(val, amount);

else continue;

}

}

else if (mode2 == 2) {

int year = -1;

while (true) {

cout << endl;

cout << "Enter the Year or Enter 0 to go back" << endl;

cin >> year;

if (year == 0)break;

cout << endl;

cout << "Enter the number of games you want" << endl;

cin >> amount;

cout << endl;

cout << "Enter 1 to get Best games by sales and 2 to get Best Games by User Score" << endl;

cin >> mode;

cout << endl;

if (mode == 1)yearArray->yearSales(year, amount);

else if (mode == 2) yearArray->yearScore(year, amount);

else continue;

cout << endl;

}

}

else if (mode2 == 3) {

while (true) {

cout << endl;

cout << "Enter the Genre or Enter y to go back" << endl;

cin.ignore();

getline(cin, val);

if (val == "y")break;

cout << endl;

cout << "Enter the number of games you want" << endl;

cin >> amount;

cout << endl;

cout << "Enter 1 to get Best games by sales and 2 to get Best Games by User Score" << endl;

cin >> mode;

cout << endl;

if (mode == 1)genreTable->genreSales(val, amount);

else if (mode == 2) genreTable->genreScore(val, amount);

else continue;

}

}

else if (mode2 == 4) {

while (true) {

cout << endl;

cout << "Enter the Age Rating or Enter y to go back" << endl;

cin.ignore();

getline(cin, val);

if (val == "y")break;

cout << endl;

cout << "Enter the number of games you want" << endl;

cin >> amount;

cout << endl;

cout << "Enter 1 to get Best games by sales and 2 to get Best Games by User Score" << endl;

cin >> mode;

cout << endl;

if (mode == 1)ratingArray->ratingSales(val, amount);

else if (mode == 2) ratingArray->ratingScore(val, amount);

else continue;

}

}

else if (mode2 == 5) {

while (true) {

cout << endl;

cout << "Enter the name of the developer or Enter y to go back" << endl;

cin.ignore();

getline(cin, val);

if (val == "y")break;

cout << endl;

cout << "Enter the number of games you want" << endl;

cin >> amount;

cout << endl;

cout << "Enter 1 to get Best games by sales and 2 to get Best Games by User Score" << endl;

cin >> mode;

cout << endl;

if (mode == 1)developerTable->developerSales(val, amount);

else if (mode == 2) developerTable->developerScore(val, amount);

else continue;

}

}

else if (mode2 == 6) {

while (true) {

cout << endl;

cout << "Enter the name of the Publisher or Enter y to go back" << endl;

cin.ignore();

getline(cin, val);

if (val == "y")break;

cout << endl;

cout << "Enter the number of games you want" << endl;

cin >> amount;

cout << endl;

cout << "Enter 1 to get Best games by sales and 2 to get Best Games by User Score" << endl;

cin >> mode;

cout << endl;

if (mode == 1)publisherTable->publisherSales(val, amount);

else if (mode == 2) publisherTable->publisherScore(val, amount);

else continue;

}

}

else continue;

}

}

else if (mode == 3) {

while (true) {

cout << endl;

cout << "Enter the name of the Game or Enter y to go back" << endl;

cin.ignore();

getline(cin, val);

if (val == "y")break;

cout << endl;

cout << "Enter the number of games you want" << endl;

cin >> amount;

cout << endl;

cout << "Enter 1 to get Best related games according to sales and 2 to get Best related Games according User Score" << endl;

cin >> mode;

cout << endl;

if (mode == 1)gamesgraph->relatedGamesSales(val, amount);

else if (mode == 2) gamesgraph->relatedGamesScores(val, amount);

else continue;

}

}

else if (mode == 4) {

cout << endl;

cout << "Press the associated number to select one of the following option:" << endl << "1. Get the Games with the best User Score" << endl;

cout << "2. Get the Games with the Higest Sales" << endl << "3. Get the games with the Highest Critic Score" << endl << "4. Get the Games with the Highest user Count" << endl;

cout << "Enter 0 to go back" << endl;

cin >> mode2;

if (mode2 == 0)break;

cout << endl;

if (mode2 == 1) {

while (true) {

cout << endl;

cout << "Enter the number of games you want or Enter 0 to go back" << endl;

cin >> amount;

if (amount == 0)break;

cout << endl;

userScoreList->print(amount);

}

}

else if (mode2 == 2) {

while (true) {

cout << endl;

cout << "Enter the number of games you want or Enter 0 to go back" << endl;

cin >> amount;

if (amount == 0)break;

cout << endl;

salesList->print(amount);

}

}

else if (mode2 == 3) {

while (true) {

cout << endl;

cout << "Enter the number of games you want or Enter 0 to go back" << endl;

cin >> amount;

if (amount == 0)break;

cout << endl;

criticScoreList->print(amount);

}

}

else if (mode2 == 4) {

while (true) {

cout << endl;

cout << "Enter the number of games you want or Enter 0 to go back" << endl;

cin >> amount;

if (amount == 0)break;

cout << endl;

userCountList->print(amount);

}

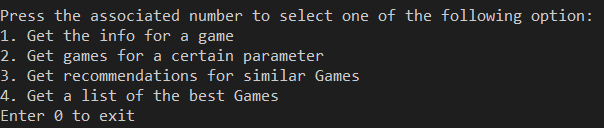
}

}

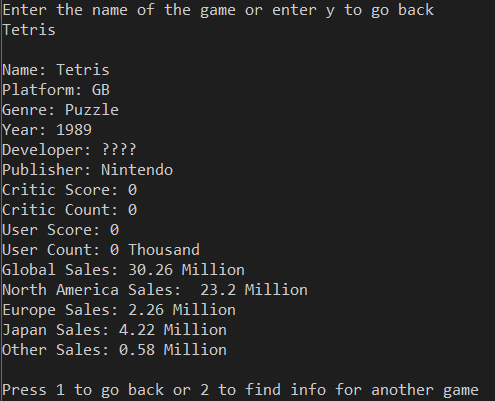
else continue;

}

}



Here the first choice will give us the info for a game.



Here is the code for this function:

void GamesGraph::info(string name) {

GamesGraphNode\* val=search(name);

if (val == NULL) {

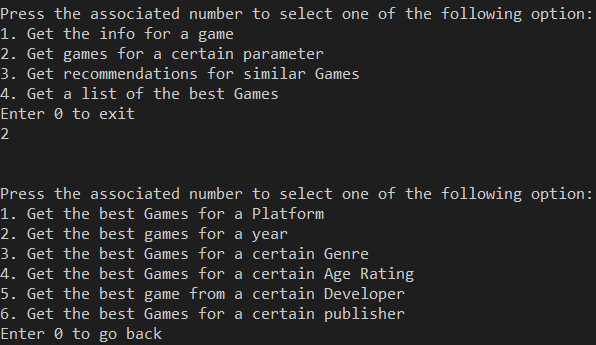
cout << "Game not found " << endl;

return;

}

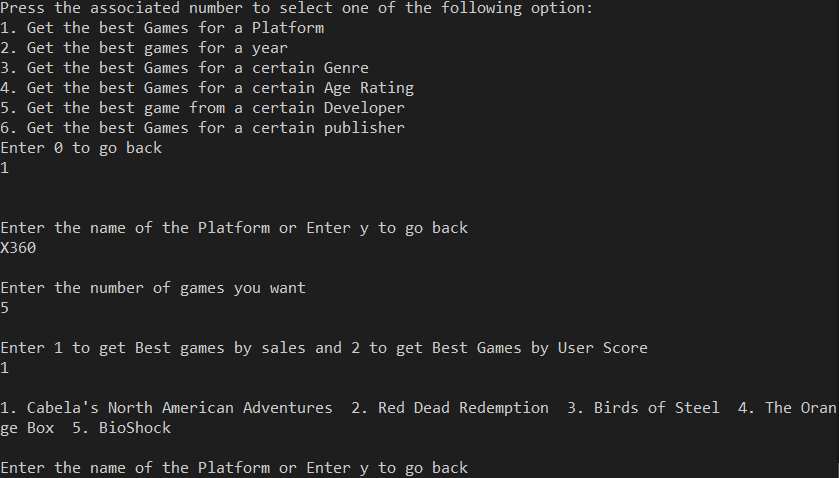
cout << "Name: " << val->name << endl << "Platform: " << val->game->platform << endl << "Genre: " << val->game->genre << endl << "Year: " << val->game->year << endl << "Developer: " << val->game->developer << endl << "Publisher: " << val->game->publisher << endl << "Critic Score: " << val->game->criticScore << endl << "Critic Count: " << val->game->criticCount << endl << "User Score: " << val->game->userScore << endl << "User Count: " << val->game->userCount << " Thousand" << endl << "Global Sales: " << val->game->globalSales->sale << " Million" << endl << "North America Sales: " << val->game->salesNA << " Million" << endl << "Europe Sales: " << val->game->salesEU << " Million" << endl << "Japan Sales: " << val->game->salesJP << " Million" << endl << "Other Sales: " << val->game->salesOther <<" Million"<< endl;

}



Choosing 2 to get games by perimeters will further give us 6 more options.

Pressing 1 will give us games by platform.



We can further press 1 or two to get games by highest sales of highest score. The code for these two functions is as follows.

void platformSales(string val,int amount) {

ListNode \*index=retrieveElement(val)->salesList->start;

if (index == NULL) {

cout << "Game not found " << endl;

return;

}

for (int i = 1; i <= amount; i++) {

cout <<i<<". "<< index->game->name->name << " ";

if (index->next == NULL) {

cout << "The list of games has been exhausted";

break;

}

else index = index->next;

}

cout << endl;

}

void platformScore(string val, int amount) {

ListNode\* index = retrieveElement(val)->scoresList->start;

if (index == NULL) {

cout << "Game not found " << endl;

return;

}

for (int i = 1; i <= amount; i++) {

cout << i << ". " << index->game->name->name << " ";

if (index->next == NULL) {

cout << "The list of games has been exhausted";

break;

}

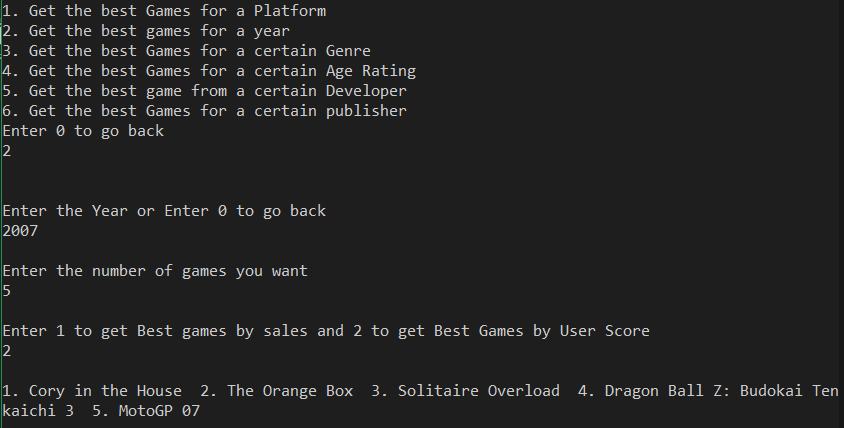
else index = index->next;

}

cout << endl;

}

Going back to the previous menu, we can press 2 to get best Games by year which again can be further filtered by highest sales and highest score.



Here is the relevant code:

void yearSales(int val, int amount) {

ListNode\* index = getLists(val)->salesList->start;

if (index == NULL) {

cout << "Game not found " << endl;

return;

}

for (int i = 1; i <= amount; i++) {

cout << i << ". " << index->game->name->name << " ";

if (index->next == NULL) {

cout << "The list of games has been exhausted";

break;

}

else index = index->next;

}

cout << endl;

}

void yearScore(int val, int amount) {

ListNode\* index = getLists(val)->scoresList->start;

if (index == NULL) {

cout << "Game not found " << endl;

return;

}

for (int i = 1; i <= amount; i++) {

cout << i << ". " << index->game->name->name << " ";

if (index->next == NULL) {

cout << "The list of games has been exhausted";

break;

}

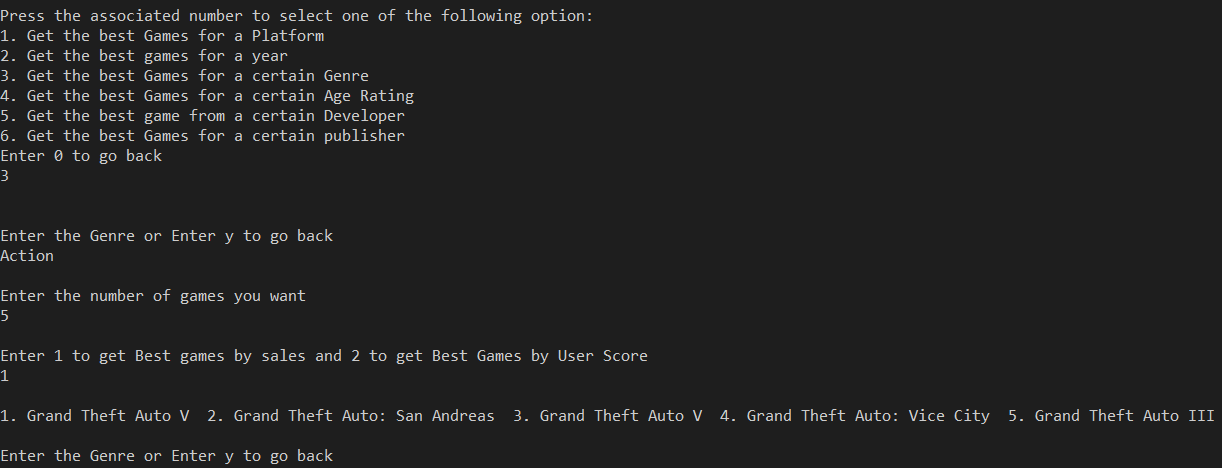
else index = index->next;

}

cout << endl;

}

Going back to the previous menu, we can press 3 to get best Games by a certain genre which again can be further filtered by highest sales and highest score.



The relevant code is as follows:

void genreSales(string val, int amount) {

ListNode\* index = retrieveElement(val)->salesList->start;

if (index == NULL) {

cout << "Game not found " << endl;

return;

}

for (int i = 1; i <= amount; i++) {

cout << i << ". " << index->game->name->name << " ";

if (index->next == NULL) {

cout << "The list of games has been exhausted";

break;

}

else index = index->next;

}

cout << endl;

}

void genreScore(string val, int amount) {

ListNode\* index = retrieveElement(val)->scoresList->start;

if (index == NULL) {

cout << "Game not found " << endl;

return;

}

for (int i = 1; i <= amount; i++) {

cout << i << ". " << index->game->name->name << " ";

if (index->next == NULL) {

cout << "The list of games has been exhausted";

break;

}

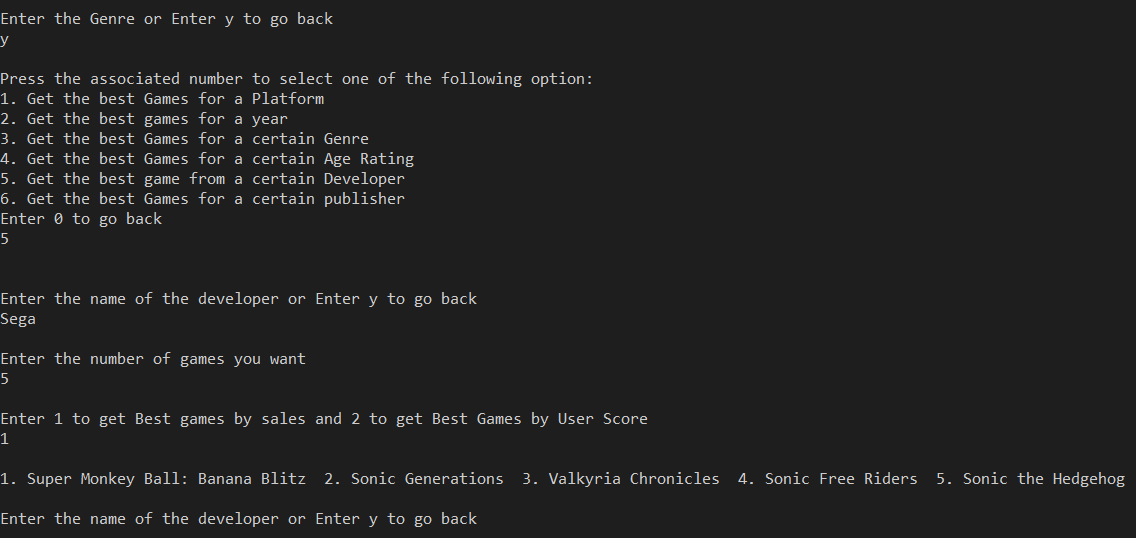
else index = index->next;

}

cout << endl;

}

We can also get the best games from a certain developer by pressing 5.



void developerSales(string val, int amount) {

ListNode\* index = retrieveElement(val)->salesList->start;

if (index == NULL) {

cout << "Game not found " << endl;

return;

}

for (int i = 1; i <= amount; i++) {

cout << i << ". " << index->game->name->name << " ";

if (index->next == NULL) {

cout << "The list of games has been exhausted";

break;

}

else index = index->next;

}

cout << endl;

}

void developerScore(string val, int amount) {

ListNode\* index = retrieveElement(val)->scoresList->start;

if (index == NULL) {

cout << "Game not found " << endl;

return;

}

for (int i = 1; i <= amount; i++) {

cout << i << ". " << index->game->name->name << " ";

if (index->next == NULL) {

cout << "The list of games has been exhausted";

break;

}

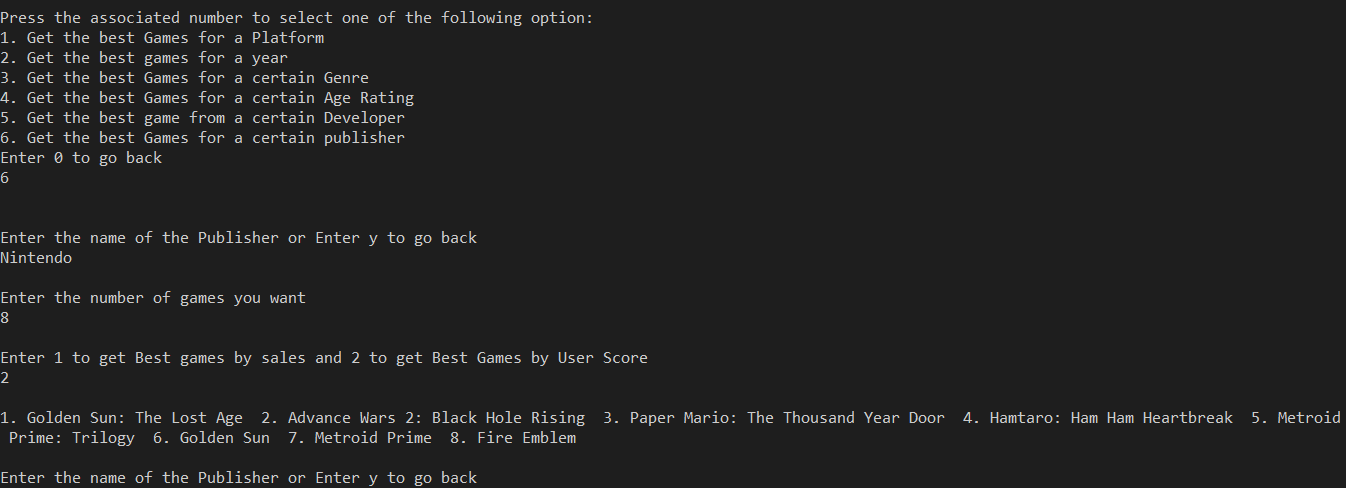
else index = index->next;

}

cout << endl;

}

Also, we can get the best games by a certain publisher by pressing 6.



The relevant code is as follows:

void publisherSales(string val, int amount) {

ListNode\* index = retrieveElement(val)->salesList->start;

if (index == NULL) {

cout << "Game not found " << endl;

return;

}

for (int i = 1; i <= amount; i++) {

cout << i << ". " << index->game->name->name << " ";

if (index->next == NULL) {

cout << "The list of games has been exhausted";

break;

}

else index = index->next;

}

cout << endl;

}

void publisherScore(string val, int amount) {

ListNode\* index = retrieveElement(val)->scoresList->start;

if (index == NULL) {

cout << "Game not found " << endl;

return;

}

for (int i = 1; i <= amount; i++) {

cout << i << ". " << index->game->name->name << " ";

if (index->next == NULL) {

cout << "The list of games has been exhausted";

break;

}

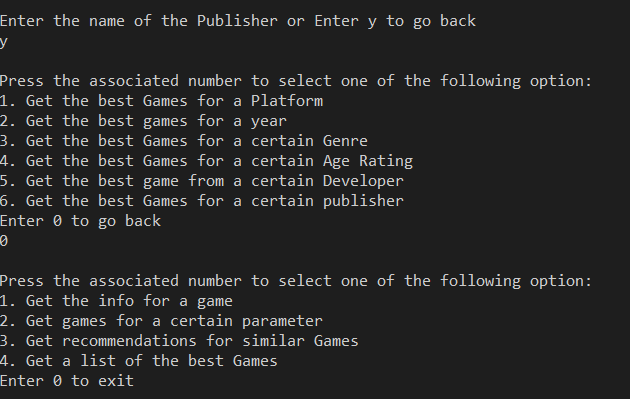
else index = index->next;

}

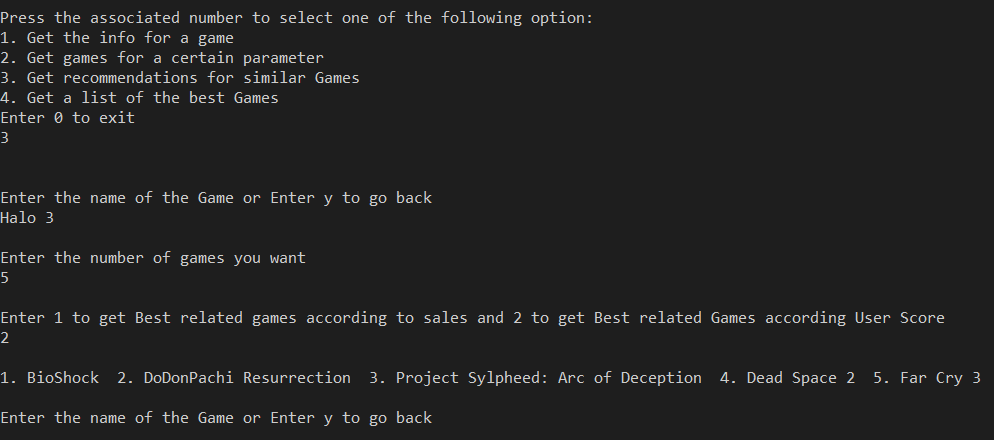
cout << endl;

}

Going back to the main menu:



We can press 3 to get recommendations for similar games and for that we have used graphs.



These recommendations can also be suggested according to highest sales or scores.

Following is the relevant code:

void GamesGraph::relatedGamesSales(string name,int amount) {

ListNode \*index=search(name)->relatedGamesSales->start;

if (index == NULL || index->next == NULL) {

cout << "Game not found " << endl;

return;

}

for (int i = 1; i <= amount; i++) {

cout << i << ". " << index->game->name->name << " ";

if (index->next == NULL) {

cout << "The list of games has been exhausted";

break;

}

else index = index->next;

}

cout << endl;

}

void GamesGraph::relatedGamesScores(string name, int amount) {

ListNode\* index = search(name)->relatedGamesScores->start;

if (index == NULL || index->next==NULL) {

cout << "Game not found " << endl;

return;

}

index = index->next;

for (int i = 1; i <= amount; i++) {

cout << i << ". " << index->game->name->name << " ";

if (index->next == NULL) {

cout << "The list of games has been exhausted";

break;

}

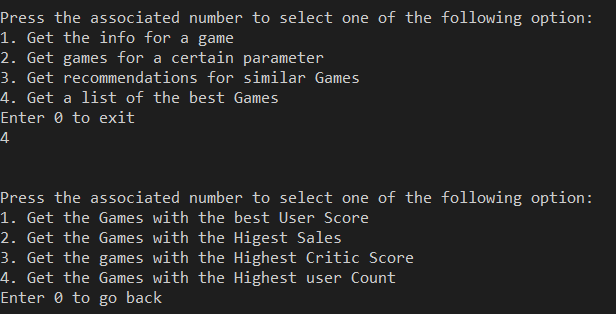
else index = index->next;

}

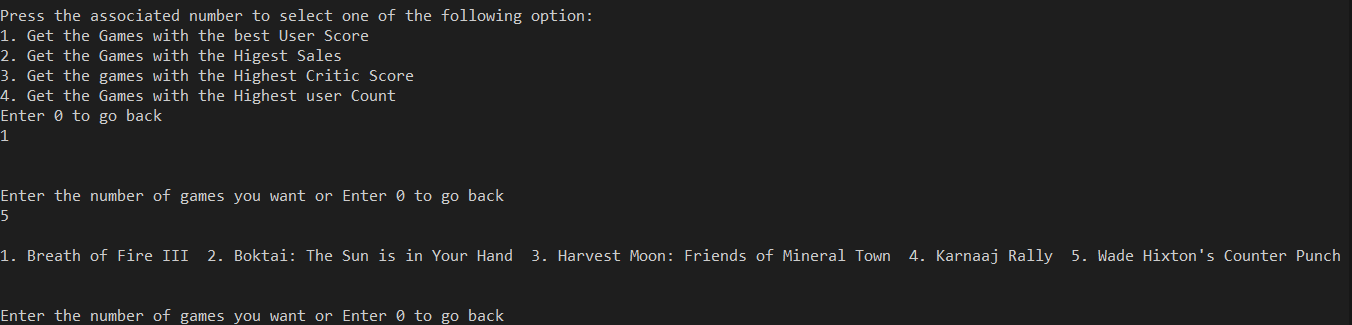
cout << endl;

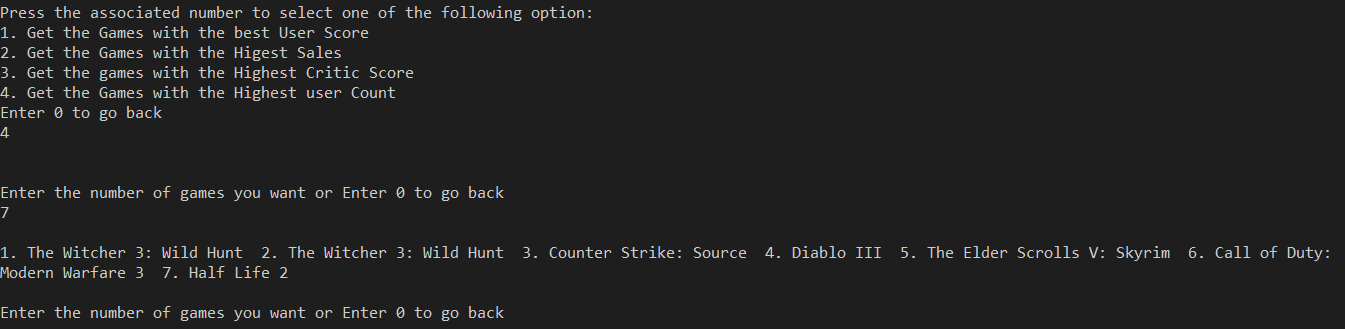
}

Going back to the main menu, we can press 4 to get the best games of all time with respect to a certain criteria:



As seen here, we can make four different further choices.





Here is the relevant code:

void print(int amount) {

ListNode\* index = start;

for (int i = 1; i <= amount; i++) {

cout << i << ". " << index->game->name->name << " ";

if (index->next == NULL) {

cout << "The list of games has been exhausted";

break;

}

else index = index->next;

}

cout << endl;

}

This code combined with following mode function can create 4 different functionalities:

bool search(Game\* game) {

if (start != nullptr) {

ploc = nullptr;

loc = start;

if (mode == 0) {

while (loc != nullptr && game->userScore < loc->game->userScore) {

ploc = loc;

loc = loc->next;

}

}

else if (mode == 1) {

while (loc != nullptr && game->globalSales->sale < loc->game->globalSales->sale) {

ploc = loc;

loc = loc->next;

}

}

else if (mode == 2) {

while (loc != nullptr && game->criticScore < loc->game->criticScore) {

ploc = loc;

loc = loc->next;

}

}

else if (mode == 3) {

while (loc != nullptr && game->userCount < loc->game->userCount) {

ploc = loc;

loc = loc->next;

}

}

if (loc == nullptr) return false;

else if (loc->game == game) return true;

else return false;

}

else { return false; }

}

void insertSorted(Game\* game) {

if (!search(game)) {

if (ploc == nullptr) { insertAtFront(game); }

else if (loc == nullptr) { insertAtEnd(game); }

else {

ListNode\* newNode = new ListNode(ploc, game, loc);

ploc->next = newNode;

loc->previous = newNode;

}

}

}