JUMBLE GAME REPORT

Brad Lyons

S5176965 bradley.lyons@griffithuni.edu.au

Contents

Problem Statement

The goal is to create a terminal game in C that is similar to “Bejeweled” or “Candy Crush”. The game is played on a n*n size board, where n is a user input. The user will enter coordinates for the *from* location and the *to* location. These two tiles will be swapped if the move is valid, otherwise the swap will not occur. If the move creates three or more tiles where the symbol matches horizontally or vertically, the symbols will vanish, and 10 points will be added to the user’s score for each symbol that disappeared. The now empty space will be filled with any symbols above it and the empty spaces that are at the to of the board will be populated with new randomized symbols.

Requirements Section

User Requirements

The user requirements are:

- The board size will be decided by the user.
- The user will nominate the number of symbols at the start of a game.
- The user will be able to enter coordinates to modify the board.
- The user will be able to save and load a game state.
- The user will be able to quit the game at any time.

Software Requirements

1. Board size is n*n, where n is a user input.
2. The number of symbols will be decided by the user.
3. The symbol pool will be chosen at random.
4. If three or more symbols are vertically or horizontally adjacent, they are removed, points are added, and a gravity effect happens. The remaining empty spaces are filled with new symbols.
5. Each symbol removed counts for 10 points.
6. The game is played against a clock. The final score is equal to the quotient of the total points divided by the time in seconds.
7. Combos are permitted. The user should never see a board with a match already on it.
8. A swap that will not create a match is not permitted and will be rejected.
9. If a move is successful, the updated board will be displayed along with the points gained and the cumulative score.
10. The game must have functionality to save and load a game state.
11. The user must be able to quit the game.

Software Design

Program I/O

Input

In the main menu, there will be a character input for a new game or a load game. If a load is chosen, the user will be put into the game.
In the case of a new game, the user will be prompted for the details of the new game, then put into the newly generated game.
Whilst in game, the user will input a character to select a command (Save/Enter Coordinates/Quit)

If the user is entering coordinates, they will enter four integers separated by spaces.

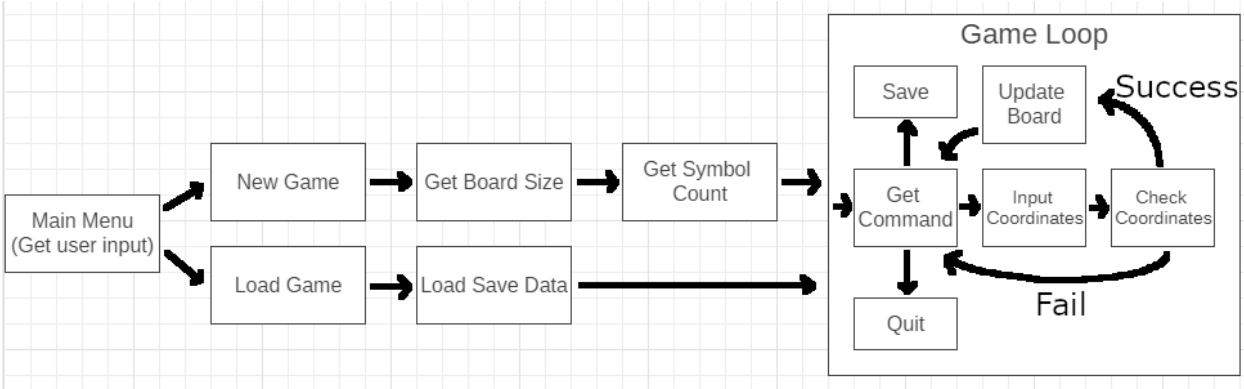
Output

The program will display a main menu on startup. Then prompting depending on the user input.
Once in game, A game board will be displayed and prompting for the users next input.

Data Section

setupData is a struct for housing the main data of the game board. It is frequently used and is created in the main function. It is constantly changed and referred to. The game board is created in main. It is a 2d array and the size is defined in the variable created from setupData. The game score is also created and lives in the main function. It is constantly changed and read from by passing its memory location as a pointer to functions that alter it.

High Level Design



Functions Section

Main()

This is the main function. The program starts here. It has an integer return type.

gameLoop()

This is the main game loop. The game data is passed to it and the function will loop infinitely until it receives a command from the user to stop, in which case it will return back to the main function where the main menu will be presented. Its input parameters are: **setupData s**, which is used for the setup data of the game. **Char board[][]** which is the game board, its size is s.boardSize. **char symbols[]**, which is a char array of the symbols used in the board and finally **int *pScore**, which is a pointer to the game score.

mainMenu()

The main menu function is for the setup of the game. It prompts the user for game details and will provide the information to the game loop once finished. It's only parameter is **s**, which is a pointer to the setupData struct that was created for the current game.

generateBoard()

The purpose of Generate Board is just to create a game board for a new game. It takes **setupData s**, **char board[][]**, and **char symbols[]** and uses them to generate a board that has no current matches.

printBoard()

Print board is a function to print the board to the terminal. It takes **setupData s** and **char b[][]**. Which is all the data it needs to print the board to the screen with a series of for loops. It has two versions it will print depending on the boardSize. One is for single digit board sizes and the other is for anything over single digit.

genSymbolPool()

This function simply populates the array for the symbols of the size that the user specified with random symbols. It used srand(time(0)) to generate a random number then checks if that has been generated. Once the array has been filled. It stops. The input parameters are: **setupData s**, **char symPool[]** and **char symbols[]**. Sympool is a constant size because it is the entire pool of possible symbols, but sym is sized according to the user specified amount of symbols.

removeMatches()

This function takes **setupData s** and **char board[][]** and removes any matches that are in the board. It does one pass for each direction on the board and flags any matches on a duplicate board with a 'd'. Then it replaces the 'd' characters with whitespace characters on the main board so that other functions can do further changing.

loadGame()

Load Game does what the title says. It takes a pointer to the game score (**int *pScore**) and reads the data on the game's save file. After reading all of the data, it will pass it into the game loop. Loadgame is called from the Main menu, so if a loaded game is quit, the chain will go backwards through functions and end up at the main menu again. (gameLoop <-> loadGame <-> main)

gravity()

This is the function used to simulate gravity. If there is any white space characters on the board, it will pull the symbols above down to fill the gap. Like most other board-altering functions, it takes **setupData s** and **char board[][]** as parameters.

okMove()

The only purpose of this function is to return 1 if a move is permissible and return 0 if it is not. It takes **setupData s**, **char board[][]** and **int coords[]** as parameters. Both s and board[][] are mentioned in nearly every function but the int array coords[] has a constant size of 4 and houses the four input coordinates.

repopulate()

Repopulate is just a series of for loops that was made into a function to make the main function look clean and be easily readable. This function just repopulates any white space with new randomized symbols from the user defined list. It is used every time the gravity function is called but was split into its own function for readability. The parameters include: **setupData s**, **char b[][]**, and **char symbols[]**.

checkBoard()

This function is similar to the Remove Matches Function, but instead of removing matches, it just checks if there are any matches and returns a 1 or 0 accordingly. It is used to check valid moves. It takes **setupData s** and **board[][]** as parameters.

calcScore()

Calcscore takes **s**, **board[][]** and the game score pointer and calculates the score by counting the whitespace on the board.

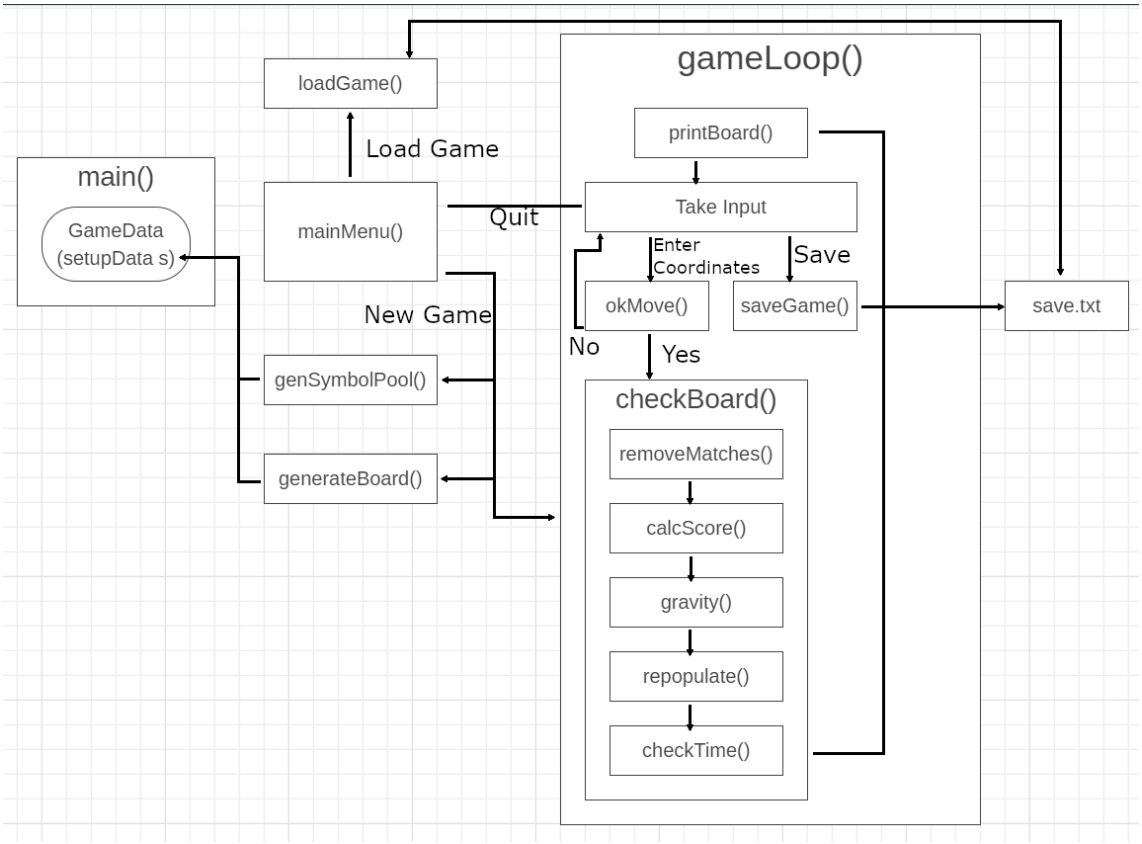
checkTime()

The only job this function has is to take the **setupData s** pointer and the session start time (**time_t start**) and update the elapsed time in the **s** variable in main with the current time. This is a small function but is frequently used to update the game's clock so it was made into a function for readability.

saveGame()

Save game saves the game. It takes all of the possible board data as parameters and saves it to a text file in a format that is readable by the load Game function. It will return 1 if successful.

Structure Chart



Algorithms

Game Loop

Take Input
If input == q, quit

Check coord input with okMove()
If not ok, start again
If ok, perform match detection loop.

Main Menu

Take input for new or load game.
If input == load, run load function.
If input == new, get input for game data.

Generate Board

Run nested for loops (boardSize) times
Fill each cell with random symbol

Print Board

If (boardSize) < 10,
Run nested for loops for size (boardSize)
Fill column numbers with static values
Populate grid with board data
Else if (boardSize) >9,
Run nested for loops for size (boardSize)
Fill column numbers with dynamic values
Populate grid with board data

Gen Symbol Pool

Run for loop (symbolCount) times,
In infinite loop:
Generate random number in range 0-9
Check if duplicate
If not, break
Else, loop again

Remove Matches

Two nested for loops of size (boardSize)
For each cell, check if next cell is a match
If match, combo++
If not, check if combo exists
If combo is >2,
Flag tempboard for removal

If in last cell, check if combo exsists,
If exists, flag for removal
Else, clear combo

if input == s, save
if input == c, take coord input.

Load Game

For each char in save.txt
Assign appropriate value to setupData s pointer

Gravity

For loop of size (boardSize)
Count empty cells with for loop
X = empty cells
Bubble sort columns from bottom up
Count empty space from the top to bottom
If x amount is counted sequentially,
Move on.

Ok Move

In Infinite Loop,
If coords are next to each other, break
Else, return error
Create test board
Decrement each coord
swap symbols with index of coords
check board
if match found, return true
else return error

Repopulate

In nested for loops of size (boardSize),
If cell == blank, fill with random symbol

Check Board

Same logic as remove matches,
But instead of removing any matches
It will return an error instead

Calc Score

In two nested for loops of size (boardSize),
If cell == blank, score = score + 10

Check Time

Run time(&now)
Set setupData time pointer to now-startTime

Save Game

For each piece of setup data, write value to file
For each symbol on board, write to file

Test Data

Requirement Acceptance Tests

Software Req No.	Test	Implemented	Test Results	Comments
1	User selected board size	Fully	Pass	
2	User selected symbol count	Fully	Pass	
3	Random symbol pool	Fully	Pass	
4	Matches will add points, remove, then symbols fall down and board is repopulated	Fully	Pass	
5	Each removed symbol counts for 10 points	Fully	Pass	
6	Game has clock and final score	Fully	Pass	
7	Combos work and user will never see board with matches	Fully	Pass	
8	A move will not be accepted if a match wont be made	Fully	Pass	
9	Successful moves will display updated board and print the score gained and total score	Fully	Pass	
10	Load and Save functionality	Fully	Pass	
11	Quit Functionality	Fully	Pass	

User Instructions

Startup

1. Once launching game, follow on screen instructions to enter a key for a new game or to load a game. After entering a key, press enter.
2. If you selected load game, your saved game will load and you can continue playing immediately.
3. In the case of a new game, you will be asked for a board size and the amount of symbols to use. The recommended board size is 5-9 and the less symbols you choose, the easier the game will be.

Playing the game

Once playing, the game will ask for a command, enter c to input coordinates to swap the tiles. You then must enter the coordinates, enter the two pairs of numbers (x/y or top number the left number) into the console then press enter. They must be separated by a space. The game will only accept moves that make a match and if you make an invalid move, you will receive a message.

You are being timed, and your end score will depend on how quickly you made the matches. You will receive 10 points per symbol matched, and your final score will be your total score/time played. To end the game, simply press q as a command.

Saving and Loading

To save your progress, just press s as a command, your game will be saved.

You must be in the main menu to load a game, just press ‘L’ when prompted.