Final Project

Data Structures CS2001

Xonix Game

Submitted to: **Ma'am Naveen Khan**

Submitted by:

Hanaa Sajid i242029

Wajiha Abbasi i242029

Date of Submission:

30th November 2025

Section:

CY-B

TABLE OF CONTENTS

1. Introduction

Our project implements a basic version of the XONIX game with many additional features like login, multiplayer, matchmaking, theme selection, points tracking, friend request, save game, and leaderboard. We used C++ and SFML library.

2. Work Distribution

Main menu	Hanaa Sajid
Login/Authentication	Wajiha Abbasi
Points tracking	Hanaa Sajid
Multiplayer	Hanaa Sajid
Leaderboard	Hanaa Sajid
Inventory (Themes)	Wajiha Abbasi
Friend Requests	Wajiha Abbasi
Matchmaking/Game room	Hanaa Sajid
Save Game	Wajiha Abbasi

3. Workflow Implementation Timeline

Initially we sat together to configure SFML a month before the project, after that we divided the work and first implemented the menu with dummy functions and created files for all the functions for a basic skeleton structure of the project. Then we gradually began implementing functionalities in parallel. Login/authentication was developed in parallel with points scoring, then the inventory system was developed in parallel with multiplayer mode. System audit logging was continued alongside. We integrated these modules and resolved the merge conflicts. We also tested the flows to make sure everything was working smoothly

After that leaderboard and friend requests module was developed in parallel, a lot of merge conflicts were arising as we tried to maintain the Player object consistently across modules. A difficulty level module was also implemented during this time. Also the profile storing system was developed. Finally the matchmaking/game room functionality was implemented in parallel with Save game functionality. Then we again integrated our code and sat together to thoroughly review and test all the functionalities

4. Data Structures

Minheap was used for the leaderboard. Basically the working of the leaderboard is that every time a user's points are increased, we have to check if he gets on the leaderboard, and we have to see if his points are greater than the minimum points on the leaderboard. For this we need to quickly access the minimum leaderboard points which we can do easily using the root of the minheap, this is why minheap is ideal for this use case.

Maxheap was used for constructing the priority queue. We can implement the priority queue using linked list as well but maxheap is a more efficient way of fetching the highest-scoring player so that was the method used to implement the priority queue, this is most useful for matchmaking, as we have to pair the players with the highest scores (priority) together. And priority queue allows us to quickly dequeue highest priority.

Arrays were used in many places particularly loading data from files and storing it, eg player profiles, etc, also they were used in gameplay for enemies, player freeze states, etc. both dynamic and static arrays were used, dynamic arrays were mostly used for arrays of pointers to custom objects.

AVL tree was used in the inventory system to minimize the time it would take users to find a theme. If themes were to be stored in a .txt or a linked list or array, it would take O(n) time to search for a theme by its ID. By using AVL trees, the time complexity becomes O(n log n). Also inorder traversal of AVL trees shows the themes in ascending ID order. No matter how many themes are added, the search will still be fast.

Struct was used for player objects. It was used as a central object for the entire game. It made it easier to save and load player, especially for the player profile. It was also used for storing theme object.

Linked lists were used in friend system to store each player's friends. It was also used in save game, to store game state elements . Also used in pending friend requests. I used them because they make it easier to add a new friend or request as time complexity is O(1) an they have dynamic size. Friends lists and pending requests change frequently so linked lists avoids shifting elements unlike arrays.

Hash table was used in friend system so it would be easier to find a player using username. It makes searching very fast(O(1)). The usernames are kept unique in the game so that is also ideal for hashing. So, when sending a friend request you don't need to scan hundreds of players.

Classes were used in inventory system and friend request system. They allow modularity so changes in themes would not affect other parts of the game.

File handling was used to store usernames and hashed passwords, for complete activity logging for the entire game and also for match history and notifications. It made it easier to retrieve data even after the program was restarted.

5. Challenges Faced & Solutions

A lot of big challenges were faced during the project.

1. Multiplayer scoring issues

Multiplayer scoring was already very difficult. Collisions were challenging to implement. However I still got through that somehow. But i ran into a very weird error where player 2's capturing was working normally but when player 1 captured tiles, some tiles would go to player 2 and show in his color and increase his points. I spent a lot of time debugging this and almost gave up before thinking that maybe this is happening due to the capturing logic being triggered 60 times per second, as it was happening at a rate of 60fps. Sure enough I added a bool to check whether the player had just been constructing trail (rather than blindly triggering capture whenever the player reached solid ground). This thankfully fixed the issue but the bug took a very long time to fix.

2. Leaderboard heap challenges

The leaderboard uses a minheap and we have to display the leaderboard in descending order. Also we have to sort the array only temporarily. I tried to generalise the heapify functions for this but it was not working so I ended up creating a temp array and heapifying it separately to implement the logic properly.

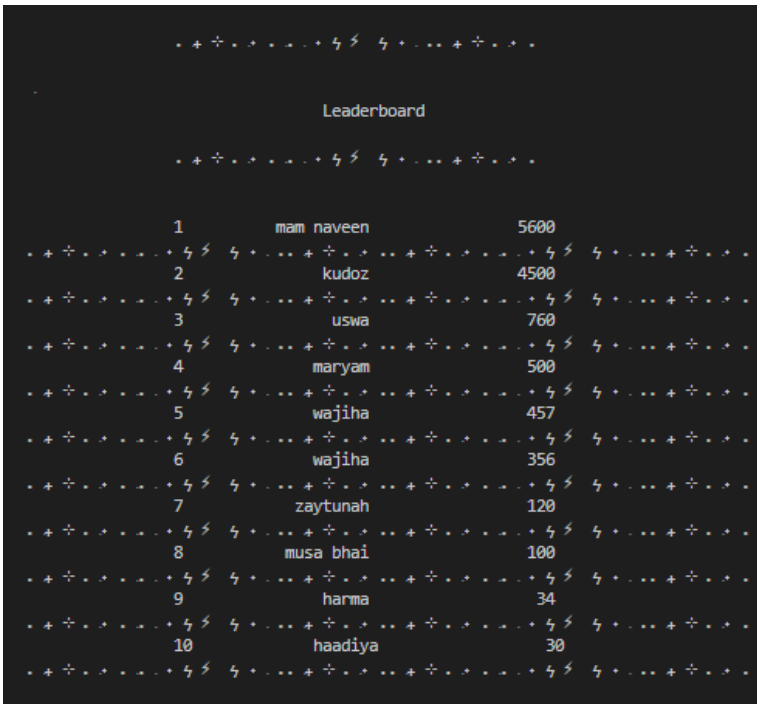
3. Problems in consistency of player object

We faced a lot of challenges in trying to keep the player object consistent once loaded. Like when the user logs in we have to load his details from the stored file and keep updating throughout play. It was hard to coordinate this across our different workflows and integrate it naturally. We had to do a lot of back and forth and discussion to properly implement it. It was used as a central object in the entire game so with every new update in the player object, other files had to be updated as well, which was very time-consuming.

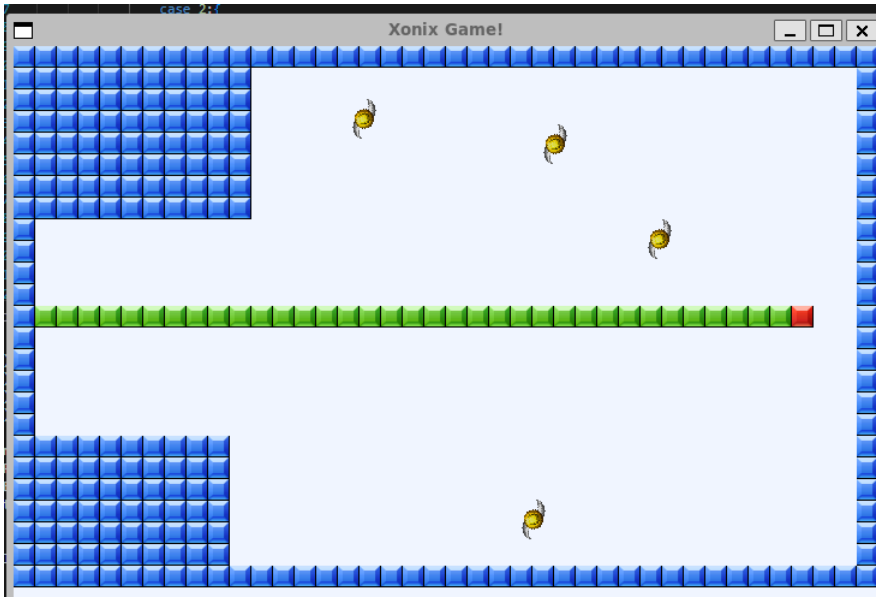
4. Problems updating login and authentication

We faced a lot of problems while updating the login function. These problems mainly arose due to a lot of merge conflicts and updating the player object frequently. I ended up remaking the login and authentication functionality over 3 times for it to work for the both of us.

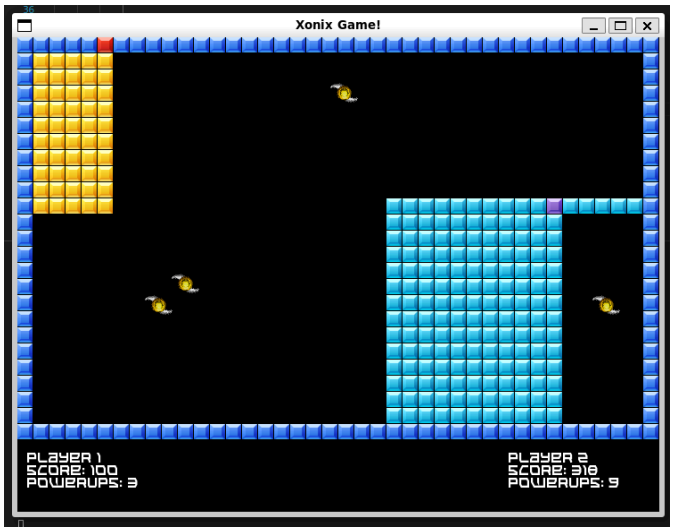
6. Screenshots & Sample output



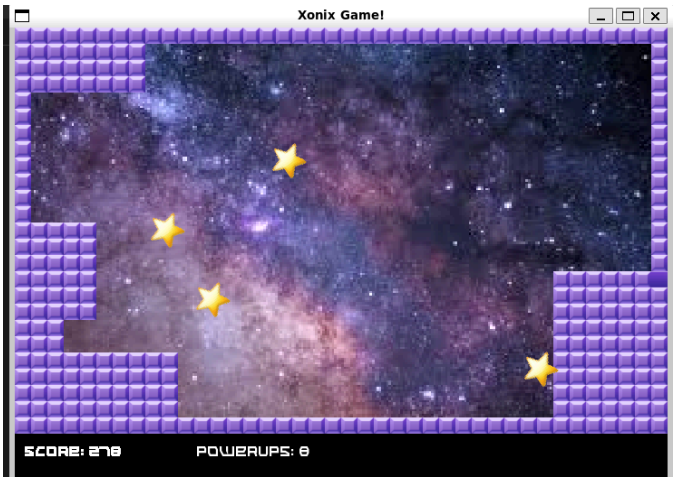
Leaderboard



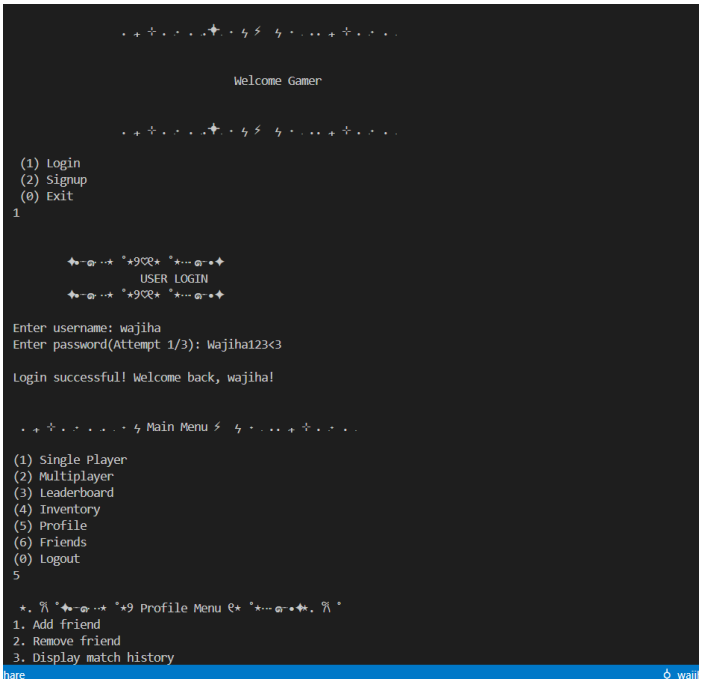
Single player

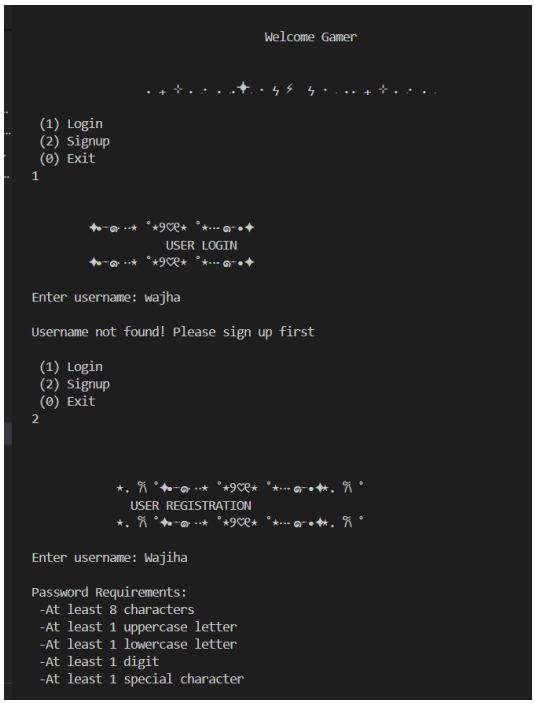


Multiplayer

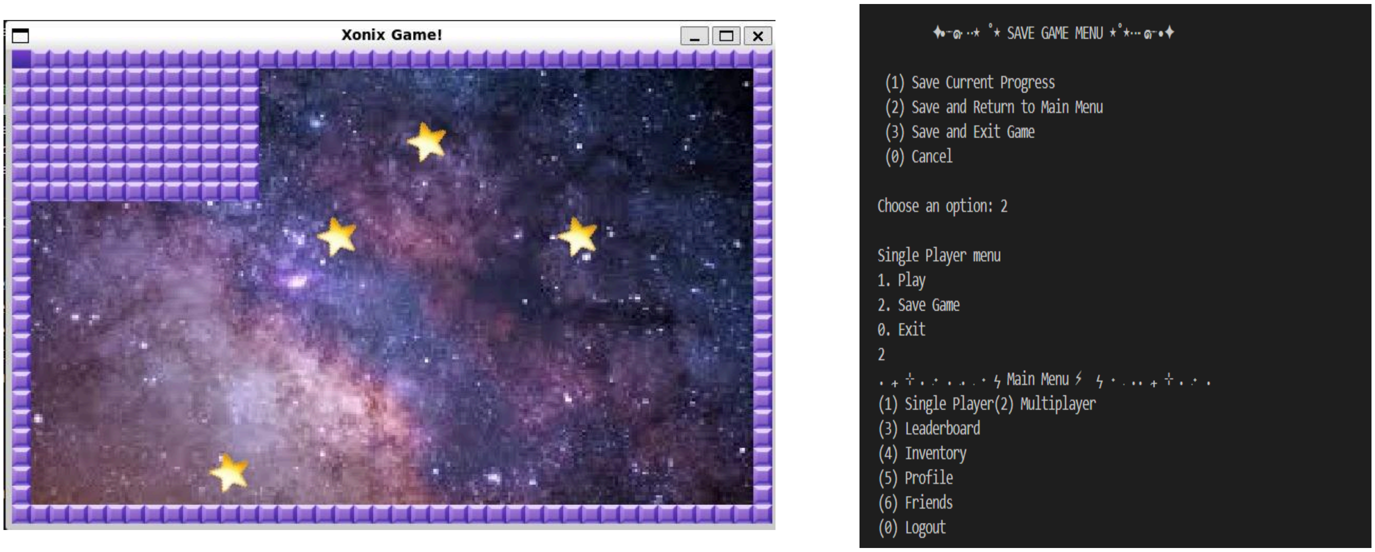


Different theme in singleplayer using inventory

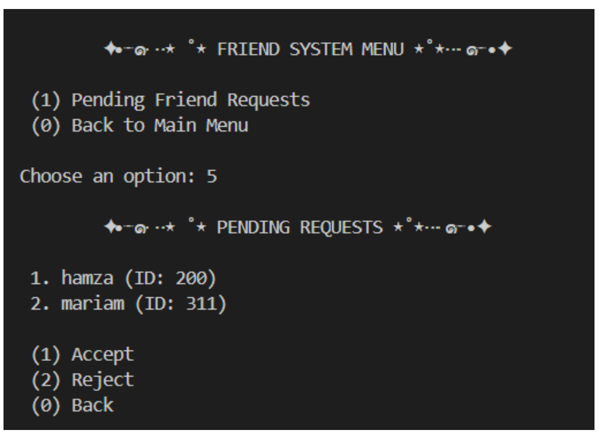




Login and authentication



Save game



Friend system menu

7. Conclusion

The project was very challenging and implemented on a time crunch, but we learned a lot from it, we learned how the data structures we studied in a theoretical way, are implemented in real world problems, we also learned about SFML library for the first time which is a very interesting tool that we can definitely explore more in the future. We also improved our coordination and teamwork too through completing challenging tasks together and helping each other resolve issues.