Online Scrabble

Software Design Document

1PI11CS109	Nikitha Nagaraj	B2
1PI11CS123	Prerana K	B2
1PI11CS137	Rakshitha K Bhat	B3
1PI11CS138	Rashmi Raghunandan	B3
1PI11CS151	Sanjana S	B3

Revision History

Revision #	Date	Description	Author	Approver	Comments
1	22/04/14	Version 1			.

Contents

1. INTRODUCTION

1.1 PURPOSE

This system design document establishes the software design and architecture of Online Scrabble which will aid software development and proceed with an understanding of how it should be built and how it is expected to be built. It captures and conveys all the significant decisions made on the architecture of the system. The intended audience for this document are Developers and Analysts, Testers and Client (In this case Professor for evaluation).

1.2 SCOPE

There are many Scrabble games available, however most games do not give the user the choice to pick a table. This way the user is at liberty to choose who he/she plays against.

The following features are in the scope of our project

1. The site offers tables, each seating a maximum of four players.
2. The identity and the number of participants in the room is made available to the user.
3. The user can choose to join a table or wait till the desired seat is vacant.
4. A user can propose an ending to the game, if other players agree the current game is ended
5. Board is visible to all players at the table and player racks are visible to the player alone.

1.3 OVERVIEW

In the following sections we outline the software product in higher detail. We will start with defining the key features that will be implemented. Next, we will discuss the constraints that will be imposed upon the software and the quality ranges, in other words, the robustness, performance, usability and reliability of the software product amongst other things. In the following sections will discuss all other product requirements, such as performance requirements and platform requirements. Next we'll outline the data description in this project. Finally individual components are explained in great detail along with test cases.

2. SYSTEM OVERVIEW

Online Scrabble offers tables for playing Scrabble with each table accommodating a maximum of 4 players. A new user will either be assigned to the next free table available or he/she can ask for a specific seat on a particular table. If this seat is available, it will immediately be assigned to the user otherwise the user must wait till that particular seat becomes free. The system will provide a list of available seats to the waiting user. As soon as the chosen seat becomes available, the user will be notified and assigned to that seat.

All users must play a game till completion or mutually agree to terminate a game. They also have an option to play again. Initially all the players will draw a single tile and the player with the highest tile will start the game. On his/her turn, the player can either pass, exchange tiles or make a play and he/she will be scored accordingly.

There will be a main process which will handle all the requests from the users. Each table will be considered to be a child process. The child process will be spawned as and when there is a requirement up till a certain threshold value. Once this value is reached, there can be no more tables generated. There will be resources allocated to each table which will be shared only by the players in that particular child process. Pipes will serve as the means to read the pool of letters. Each table will have a pipe which will be filled with alphabets in random order. The table will have write permissions on the pipe. The four players will have read and write permissions on the pipe and draw a tile as and when required. Shared memory will be used to represent the board. Arrays will serve to be the racks for each player which will not be visible to the other players.

We make use of semaphores to ensure no other player plays until a particular player has finished his/her turn.

3. SYSTEM ARCHITECTURE

3.1 ARCHITECTURAL DESIGN

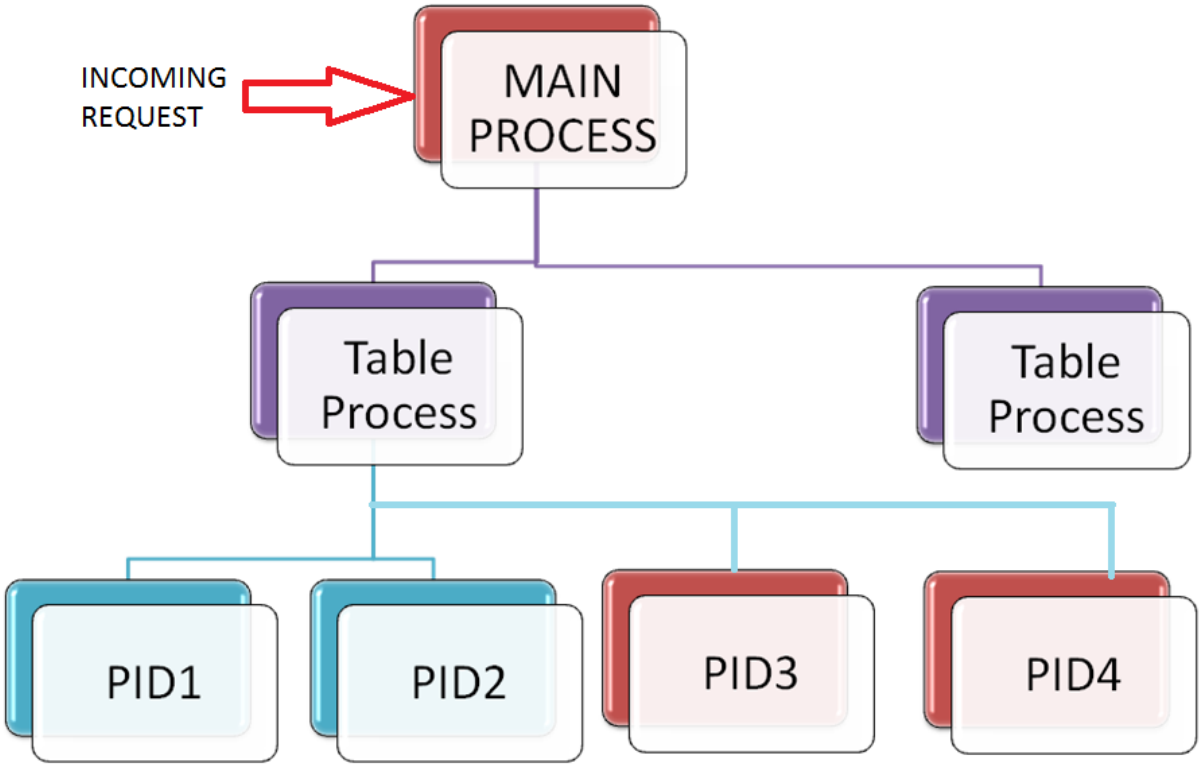


Figure 1.0 Architectural Design

NOTE: The four processes correspond to the request sent by each client.

The above image gives a brief overview of the system developed for Online Scrabble.

According to our design, every requesting client is a process having a unique PID.

The system makes use of two processes:

1. Main Process - This is the process that receives all requests. The request can be handled in five ways by the main process:
Create a new Table
Join an existing Table
Wait for any Table to become free
Wait for a particular table to become free
Wait if room is full.

2. Table Process - On response to a request, a new table may need to be created. This is a table process that will be spawned by main process. There is a limit on the number of such table processes the main table can spawn. The main process creates a pipe between itself and the table process to communicate the PID's of the processes that will play at that table.
A table process at any time can handle only four processes playing.

The IPC Resources used by our system are:

- Pipes
- Shared Memory

3.2 DESIGN RATIONALE

There are two key points that were considered when the system designers looked into design trade-offs for Online Scrabble

1. Creation of a main process vs. Letting each table independently handle the request:
   Creation of a main process which in turn would spawn one process per user is a simpler and more convenient option as opposed to having one main process per table. In the latter case, the developer would have to keep track of all the tables (number depending on the threshold set) which is a more complex design
2. Using pipes and shared memory.
   Shared memory lets multiple processes read and write which is requirement a for all four players to have access to the table. However pipes are used for IPC for passing tiles between players instead of shared memory as it is a less expensive implementation for lesser sized data transfer.

3.3 DESIGN DECOMPOSITION

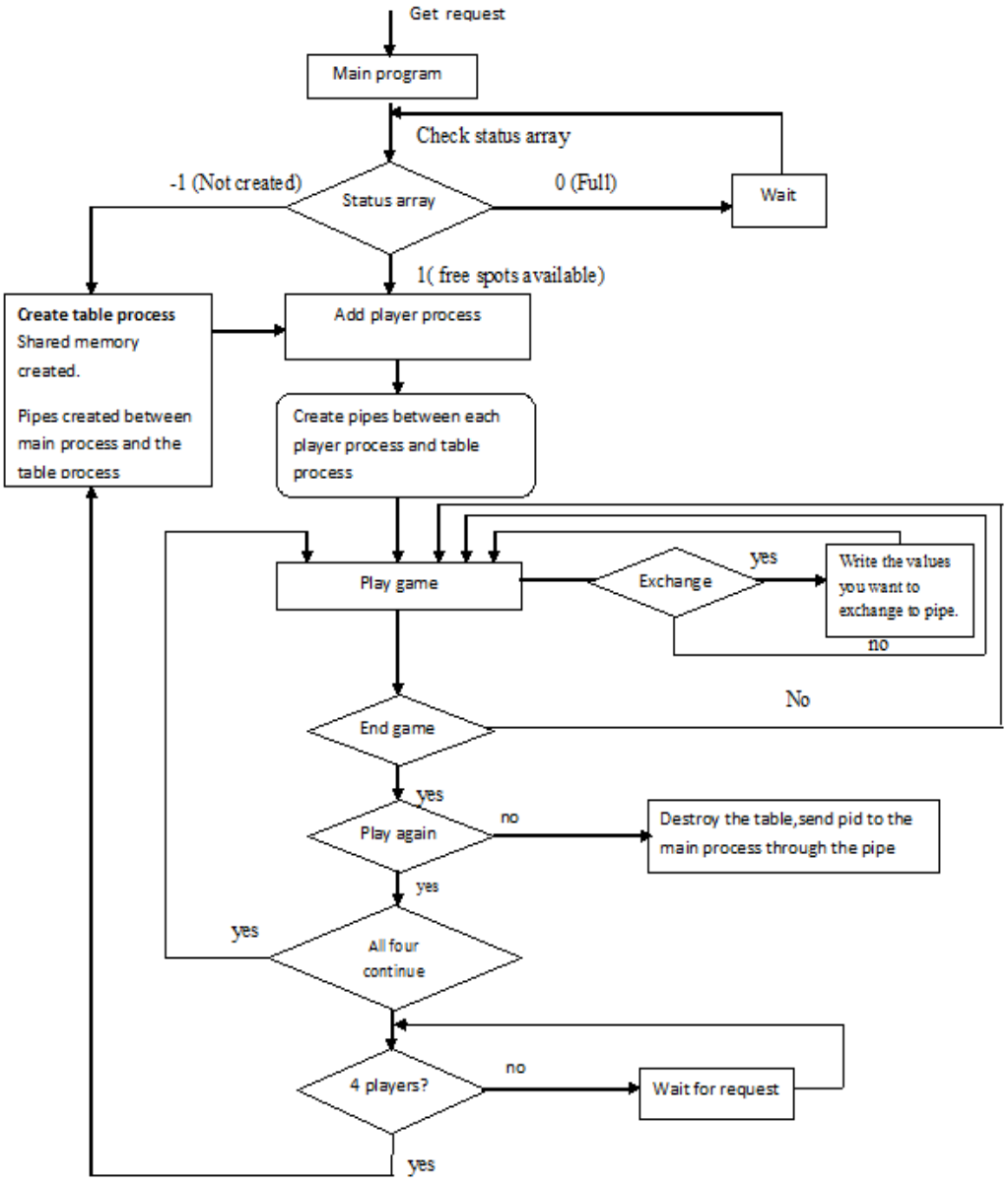
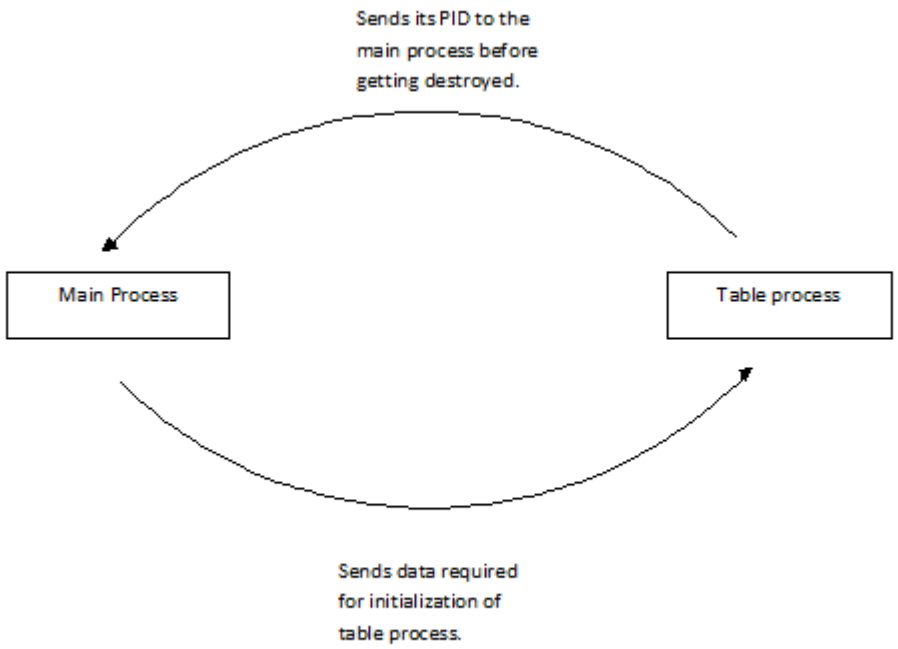
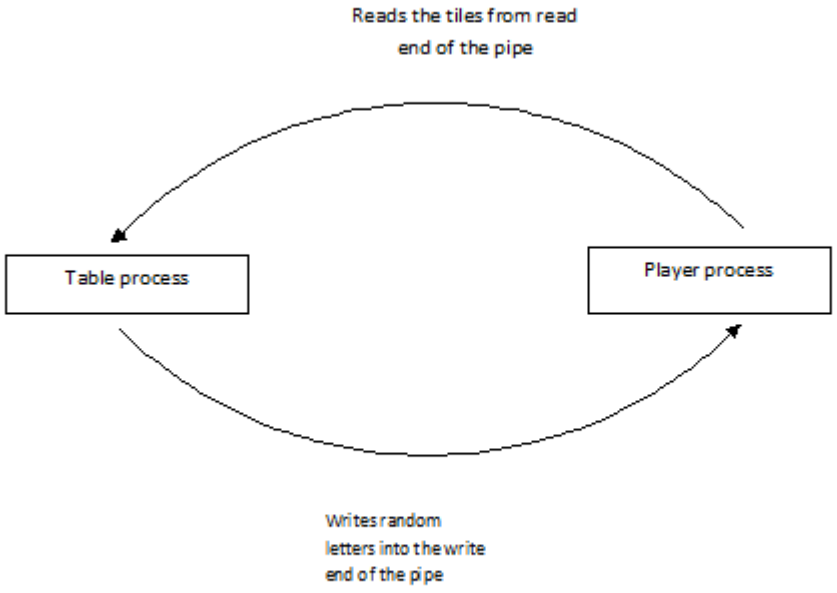


Figure 2.0 DESIGN DECOMPOSITION

Pipe between main process and the table process:



Pipe between table process and the player process:



4. DATA DESIGN

4.1 DATA DESCRIPTION

The following is the list of data structures to be used in the system:

1. Shared memory - It is created at each table. It holds the board (a two dimensional array of characters) and the score of each player. It is visible to all the players. The ID of the shared memory is given to all the players at that table. Players can read the contents of the board and the scores. They can write to the board when it is their turn. The table process updates the scores.

2. Pipes - They are created between each player process and the table process for the movement of tiles between the two processes. The table process writes the pool of letters into the pipe at the beginning of the game so that the player processes can draw tiles. The player process writes into the pipe when it wants to perform an exchange. It sends the tiles it wishes to discard. Pipes are also created between the table process and the main process for passing data required for the initialization of the table process. Data is written into this pipe by the main process. This data includes the PIDs of the processes that want to play at the table. The table writes a value into the pipe to inform the main process when a game is over.

3. Queue for waiting processes - When a player process asks to join a table, the main process may require it to wait in case the requested table is not free or all tables in the room are full. In this case, the process is put into a queue until a table becomes free. It is implemented as a priority queue. In the case where a process is at the head of the queue and has requested to join any table, but a table that was particularly requested to be joined by another process farther down the queue becomes free, the process that has requested for that table is added instead of the one at the head.

4. Status array - This contains as many elements as there are tables. The values of the elements are 1 if the table has free spots, 0 if the table is full and -1 if the table is not yet created. This is visible to all processes and is contained in the main process. This array is updated only by the main process.

5. Rack array - The tiles of the player are stored in an array in the player process. This array is viewable only by the player. It is updated by the player.

6. Pool of letters - The pool of letters from which the players must draw tiles is stored as an array at each table. A random function is called on this pool every time a player adds tiles to it for an exchange and at the beginning of the game when the players initially draw tiles. This array is updated by the table process at the beginning of the game and when each player draws a tile or adds a tile during an exchange.

5. COMPONENT DESIGN

5.1 REQUESTING TO JOIN A TABLE

Initial Condition:

1. A main process has been created to handle the requests.
2. Requests are handled one by one.
3. The main process has created all required data structures.

Function:
1. A process requests the main process to join a table.
2. The main process checks the type of request and executes one of two cases:
    Case 1: The process is ready to join any table:
        The main process checks the status of the tables.
        If no table exists or no free tables exist but the number of tables is below the limit:
            The main process creates a new table process.
        A pipe is opened between the main process and the table process.
        The PID of the requesting process is passed to the table process.
        If a table exists with free spots:
            A pipe is opened between the main process and the free table
            The PID of the requesting process is passed to the table process
        If no free tables exist and the number of tables has reached the limit:
            The process is put into the wait queue until a table becomes free.

    Case 2: The process requests to join a particular table:
        The main process checks the status of the tables.
        If the table has not yet been created:
            The main process creates a new table process.
        A pipe is opened between the main process and the table process.
        The PID of the requesting process is passed to the table process.
        If the table already exists and is free:
        A pipe is opened between the main process and the table process.
        The PID of the requesting process is passed to the table process.
    If the table exists but is not free:
        The process is put into the wait queue until the table becomes free.
3. Once the process has been passed to the table process, the main process no longer keeps track of it.

Result:
The process is either directed to an appropriate table or made to wait.

Assumptions:
All error handling is assumed to be in place. Checks for limits on the number of tables, availability of a table, request on a table number that is out of range etc. are all taken care of. The establishment of a TCP connection between the client (requesting player process) and the server (main process) and other networking aspects are not discussed in detail and assumed to be handled.

5.2 CREATION OF TABLE AND RELATED RESOURCES

Initial Condition:
As mentioned above, the main process creates a new table and associates a pipe with the table.

Function:
After the table is created, the following resources are created:
    1. Shared memory
    2.Pipe

1. Shared memory: The table process creates a shared memory which contains the double dimensional board which can be read by each of the player processes and can be written into when a player places a new word on the board. This shared memory is not available to any other table process.

2. Pipes: A pipe is created to put in the pool of letters. This pipe is read by the 4 player processes to obtain the tiles. They can write to the pipe when they want to exchange their tiles.
The table process creates these resources and attaches itself to the shared memory. The player processes also attach to the shared memory as and when they join a table.
The table process also notifies each player of the PID of the next player process which joins. The last player process to join the table, i.e the 4th player process will be provided with the PID of the first process.

Assumption:
The game does not begin until a table has four player processes. Hence the first player process must wait for 3 more requests to that table before it can start playing.

Result: The table has created all the resources required to start the game.

5.3 PLAYING THE GAME

Initial Condition:
1. Player has 7 tiles on his rack.
2. The board is completely visible to him.
3. Player is aware of the PID of the process who has his turn next.

Function:

1. If the process receives signal, signal handler is executed.
Process decrements semaphore BOARDWRITE to block other processes from writing

2. Process can:
    Case 1: Use shared memory to place correct tiles on board.
    Case 2: Put unwanted tiles into Tiles Pipe to exchange with the pool of tiles present with the table process..
    Case 3: Skip turn

3. The tiles on the rack are replenished from the function's read end of pipe so that 7 tiles are present.

4. The Process increments semaphore BOARDWRITE and blocks itself.

5. Process sends a signal to pid of next player.

Result:
Player has played his turn and passed the turn to next player.

Explanation:
To execute the condition where the player exchanges tiles, the read end of the tiles pipe is temporarily opened to input these new letters. The table process then reads data into an array, calls

random function and inserts characters into pipe again. Step 3 of the above pseudo code is executed.

5.4 END GAME

Initial Condition:
1. The rack containing all the letters is empty or All the players decide to end the game.
Function:
1. Once it is decided to end the game a choice has to be made among the following cases
Case 1: Play the game again.
Within this scenario a choice has to be made among the following cases:
Case 1:
If all the players want to replay the game:
In this case no new table needs to be created,the already created table can be reinitialized.
The function Play_game() has to be called.

Case 2:
If only few members among the players want to replay the game:
It first checks if there are any players waiting for the particular table and only when there are four players available the table will be recreated and the game is played again.
The function create_table() is called.
The function Play_game() is called.

Case 2: Destroy:
When all the players decide to end the game:
In this case the entire table is destroyed and the pid of this table process is sent as a signal to the main process from which the main process will be able to remove this game from the list of games running at the moment.
Result:
The present game has completed and either of the following situations will be a result:
1. A new game would have started with the same four players.
2. A new game would have started with one or more old players and the rest new players.
3. The table would have been destroyed completely.