

# SRS DOCUMENT

## GAME: CHAIN REACTION

### About the game:

Chain reaction is a strategy game for 2. The objective of the game is to take control of the board by eliminating the opponent's balls.

Players take it in turns to place their balls in a cell. Once a cell has reached critical mass the balls explode into the surrounding cells adding an extra ball and claiming the cell for the player. A player may only place their balls in a blank cell or a cell that contains balls of their own colour. As soon as a player loses all their balls they are out of the game.

Boxes in the corners can have only one ball. When they have more than one ball they explode.

Similarly boxes on the edges other than the corners can have a maximum of two balls. If they have more than two they explode. Boxes other than those on the edge can have a maximum of three balls so they explode when they have more than three balls.

When the balls explode into the adjacent boxes having balls of the opponents the opponent loses the balls and the balls are gained by the player.

The objective is to remove all balls of the opponents.

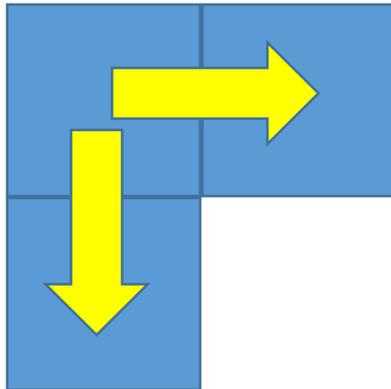
In our program we have implemented a two player version of the game.

## ALGORITHM:

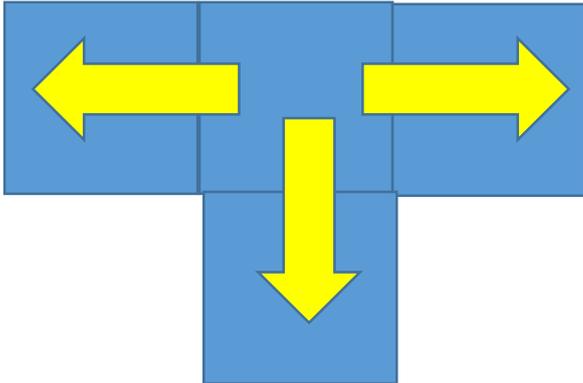
- At the start user is required to click on “ONE PLAYER” or “TWO PLAYER” to play a game versus the computer or to play with another player.
- Each time the user clicks on the box (or a box is selected by the computer at random), the number of balls in the box (box of the grid) increases by one. If the mass exceeds the critical mass, the ball explodes.
- From click input, specific box is located .User cannot select a box occupied by the opponent’s balls.
- The grid is represented by a 2 dimensional array. The array has 10 rows and 6 columns. Empty boxes have the value 0, nonempty boxes store data about player number and number of balls.
- A ball is added to the box. This is updated in the array by increasing the number of balls in the particular position of the array by 1.

- If the number of balls exceeds the maximum possible number of balls in the particular position, the ball explodes in all possible directions (only vertical and horizontal directions allowed). The number of balls in the adjacent positions of the array are incremented. The number of balls in the positions where a ball goes is increased by one.
- If after exploding any of the adjacent boxes has number of balls more than the maximum number, then this box also explodes, and the process continues recursively.
- The game terminates when all the balls of one player have been lost to the other player.
- So after each explosion or after each addition of ball checking is done to see if either of the players has won the game.
- If all the balls belong to one player then that player will win the game.
- At each stage the array is displayed using the graphics features of simplecpp.
- After completion of the game the user is given the option of viewing a replay of the game.
- The user is asked whether the game is to be played again .If the user clicks on YES the game starts again otherwise the game stops.

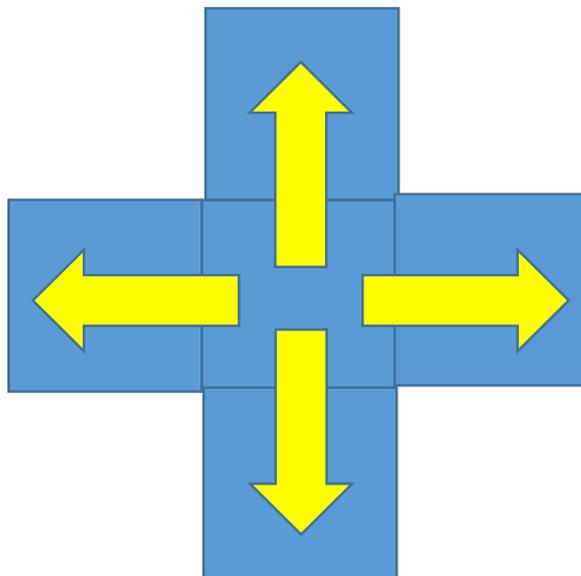
## **MOVEMENT OF BALLS FROM CORNER AFTER EXPLODING**



**MOVEMENT OF BALLS FROM BOUNDARY BOXES OTHER THAN CORNERS AFTER EXPLODING**



**MOVEMENT OF BALLS FROM BOXES IN THE INTERIOR OF THE ARRAY AFTER EXPLODING**



## **ABOUT THE CODE :**

### **FUNCTIONS AND VARIABLES:**

- **int checkWin(int \*\*grid,int player\_no)**  
This function accepts the grid .The current configuration of the grid is stored in grid 2d array. The function checks whether the player player\_no

is the winner or not .If the player `player_no` is the winner then all the balls in the grid belong to this player and the opponent does not have any balls. In this case the 2d array stores player number data as the number of the winner in all the non-empty boxes .If this condition is satisfied then the player `player_no` is the winner otherwise the player `player_no` is not the winner. This function is used to check whether the game is over or not.

- **`int isBorder(int i,int j)`**

This function checks whether position  $(i+1,j+1)$  lies on the boundary of grid or not. The function is used to check for border balls when adding balls or exploding balls.

- **`int isCorner(int i,int j)`**

This function checks whether position  $(i+1,j+1)$  is a corner box of the grid or not. The function is used to check for corner balls when adding balls or exploding balls.

- **`void add(int i,int j,int **grid,int player_no)`**

This function adds a ball to the  $(i+1,j+1)$  position of the grid .This is a ball belonging to player `player_no`.

If the box is a corner and the number of balls is more than 1 or the box is on the boundary but not on the corner and the number of balls is more than 2 or the box is in the interior and the number of

balls is more than 3 the balls in the box explode. This is implemented by the function add which recursively calls itself.

Before each call it is checked whether the game has already been won by any player .If the game is over the recursion stops and the result is declared.

- `void display(), void display(int grid[10][6])`

These 2 overloaded functions display the grid and the orbs using simplecpp graphics.

Grid is compared with 2 dimensional array (prev) to check for any changes in the positions of the grid.

- `int validateInput(int **grid,int i ,int j, int player_no)`

This function validates the input .It checks whether user has clicked inside any valid square or not.

It also checks whether position selected is inside a square occupied by opponents square or not.

- `int main()`

In main a grid is printed at the start of each new game .Input is obtained from user click or computer generated random data. Input is validated and then add () function is completed.

All data regarding the progress of the game is stored in the file “history” and each step is stored in `grid(int ** grid)` and the previous step in `prev` array. The 2 D array is displayed using `Simplecpp` graphics.