

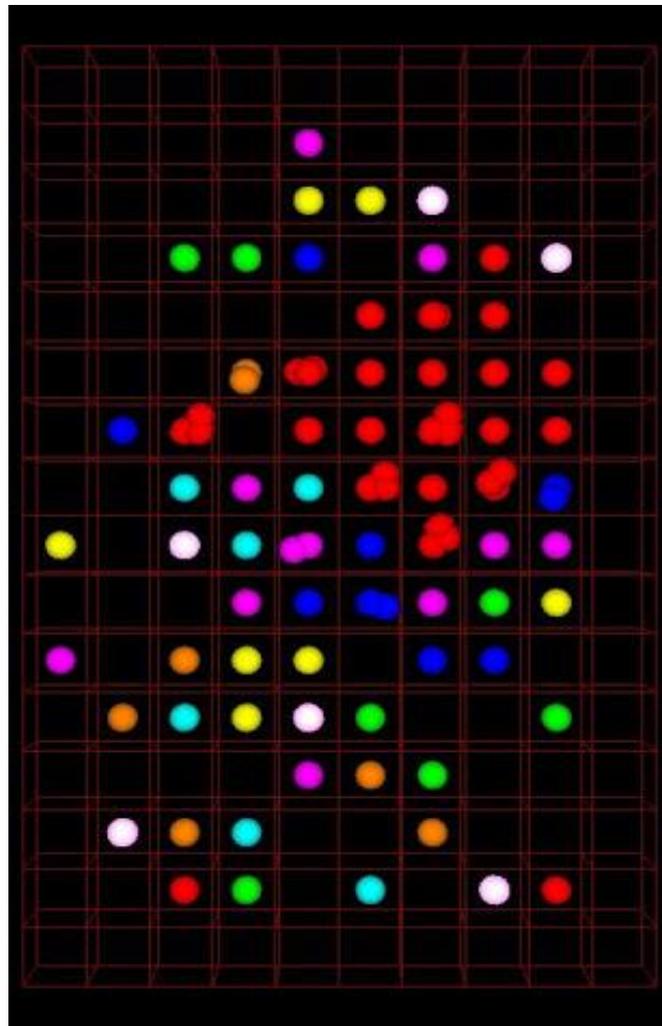
---

# PROJECT REPORT

---

PROJECT TOPIC: CHAIN REACTION

---



**ABOUT:** The objective of Chain Reaction is to take control of the board by eliminating your opponents' orbs.

## TEAM MEMBERS

---

S.No.	NAME	ROLL.NO
1.	AMIYA MAITREYA (L)	140020025
2.	ARUNABH SAXENA	140020005
3.	SHASHI KANT KUMAR	140020060

## WORK DIVISION

---

Following is the individual contributions to the project till now:

TEAM MEMBER	HOURS CONTRIBUTED	KEY CONTRIBUTIONS
1. AMIYA MAITREYA	69	❖ Researched and learnt the graphical features of SDL and coded the full graphical part of the game.

		<ul style="list-style-type: none"> <li>❖ Coded most part of the program integrating graphics and game code</li> <li>❖ Debugged all parts</li> <li>❖ Wrote the SRS</li> <li>❖ Made the algorithm on which the game is going to run.</li> <li>❖ Coded 2 possible AI codes based on Arunabh's insights.</li> </ul>
2. ARUNABH SAXENA	55	<ul style="list-style-type: none"> <li>❖ Provided ideas and insights into possible implementation for 1-player modes</li> <li>❖ Coded a better Blast() which is critical to the game.</li> <li>❖ Also helped in some game code.</li> <li>❖ Coded 1 possible AI code.</li> <li>❖ Attempted at inclusion of a code that would enable sound to play.</li> <li>❖ Wrote user manual</li> <li>❖ Wrote the Project</li> </ul>

		<p>Report</p> <ul style="list-style-type: none"><li>❖ Finalized all the documents.</li></ul>
<p>3. SHASHI KANT KUMAR</p>	<p>30.5</p>	<ul style="list-style-type: none"><li>❖ Helped play as a counter to the A.I thought -which helped him note down the fallacies possible</li><li>❖ Helped during documentation of report.</li><li>❖ Helped in creating the images of the balls, main screen etc.</li><li>❖ Coded the <code>checkWin()</code> , <code>TurnPrinter()</code>, <code>Game Over()</code></li><li>❖ Helped in debugging the code a bit.</li></ul>

---

## DESIGN AND ALGORITHM

---

We implemented our grid of the game using a 3D array named Grid [8][6][2]. This would basically give us 2 two dimensional arrays one placed behind the other. The purpose of this is to have easy access and determination of player ID. The two dimensional array [ ][ ][1] stores the ID of the player corresponding to the orb in the two dimensional array [ ][ ][0] in the front.

E.g. Assuming top right corner to be (0,0) if you take the 2,3 to have 2 balls of player with ID 3 , then Grid[2][3][1] will be equal to 3 (ID of the player) and Grid[2][3][0] will be equal to 6(number of balls \* ID). Thus whenever we require the number of balls at a location, we simply do a division!

We have kept ID = 0 to indicate that a square is unoccupied. The program gets the number of players input from the mouse click in the NumberPlayers() function. The GamePlay function is then called which is the backbone of the game.

After the grid is printed, whenever you click the mouse inside the grid we take the co-ordinate of the click using functions available in the SDL libraries. We manipulate its value by dividing by 60 and storing it as integer to give the corresponding indices of the square (60 by 60) inside which there was an event of a mouse click relative to the 8 by 6 grid we have created.

After getting the corresponding square, we edit the values in the Grid variable, ensuring the fact that an orb of a player is added only

to a square which is vacant or previously owned by the same player. After the move has been incorporated in the Grid, a call to the Blast function is invoked to check for squares which have crossed the critical limits. The Blast function has to have a recursive implementation which enables it to handle successive blasts. This is because after an explosion occurs and the orbs spread to adjacent squares the blast function needs to be invoked again so as to check for the affected squares. After a successful stable condition has been achieved with all corresponding changes made in the Grid, a call to BlitBalls function is invoked with the Grid and ID to print the corresponding pictures on the screen. After a move is done and pictures printed, the turn variable (j) is incremented by the unary operator ++. This ensures a rotation of the turn between the players. The turn of a player is intimated to him by a message adjacent to the grid which prints out the player ID of the person whose turn it is. The player colour is also the colour of the rectangle in which the player ID is flashed. Before the next iteration in the loop is started, a call to checkPlayerExist function and checkWin function is called to ensure whether the next player exists or the game has already been won! If checkWin returns a value corresponding to a player ID, GameOver function is called which prints the winning message thereby ending the game!

## SOME UNRESOLVED ISSUES...

---

1. We did not include past results as the gameplay is generally done via multiplayer option in which case we would have to store results under multiple player IDs as the history of each player who played would have to be shown. This we felt as a team was not feasible in the given timeframe.
2. The code for artificial intelligence took 3 attempts to get it right however it happened at the last moment and couldn't be

implemented in the graphic interface of the project . Thereby we are just uploading the pseudo code of the AI.

The final logic for which we settled to implement the AI was the simulation of moves to get the best possible ratio of balls in favour of the computer. What the code does is it copies the original array and calls a function known as Simulate which then using for loops simulates one possible turn of the computer followed by corresponding second turns of the player. After two moves have been simulated , the ratio of the balls of computer to the balls of the player are calculated in the final simulated grid. Likewise all the ratios are compared for all the permutations and combinations of possible computer moves and possible human moves. Thereby we get a (x,y) for which the ratio is highest. We have made the function of return type CompMove which is a structure with data members x,y and ratio. Therefore Simulate function returns a structure TheMove out of which we can extract the required move using TheMove.x and TheMove.y. Thus we get the move.

3. We also couldn't fit in the implementation of String input of names and the mouseover highlight feature in the code in the time constraint even though their non-graphical code were coded .

## ACKNOWLEDGEMENT

---

1. We would like to acknowledge the help received in this project by our mentor Sougata Singha who has been actively involved in every stage of the operation.
2. We would also like to thank our friend Harshvardhan Tibrewal with whom we had active discussions over the

possible strategy and implementation of artificial intelligence in our game.

3. We would like to thank the entire CS101 course and Professor Dr. Supratik and Professor Dr. Phatak for introducing us to the wonders of programming via these thought – provoking project ideas. It was indeed a vivid experience to see our code get a visual aspect in this project.