A MINOR PROJECT REPORT ON

ONLINE GAMING ARENA

Submitted to

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

By

Samibrata Ghosh (130203092)

Tuhin Dutta (130203120)

Shubhadeep Chowdhury (130203102)

Dipanjan Banerjee (130203039)

Sayak Chatterjee (130203098)

Bastab Dey (130203030)

Under supervision of

Asst. Prof. Madhusmita Mishra

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

JISCOLLEGE OF ENGINEERING

BLOCK 'A' PHASE 'III' KALYANI NADIA-741235

CERTIFICATE

It is certified that the work which is being presented in the B.Tech final year project (Report) entitled "**ONLINE GAMING ARENA**" submitted to the Department of computer science and engineering of JIS college of engineering by the students Samibrata Ghosh, Tuhin Dutta, Shubhadeep Chowdhury , Sayak Chatterjee, Dipanjan Banerjee, Bastab Dey is an authentic record of their own work under the supervision of Mrs.Madhusmita Mishra.

Date:
Signature of Supervisor(s)

……
Mrs. Madhusmita Mishra
Asst. Prof, CSE Department
JIS College of Engineering

Signature of HOD

……
Mr. Amrut Ranjan Jena
HOD, CSE Department
JIS College of Engineering

Contents

LIST OF FIGURES

PREFACE

As a part of project in B. Tech 4th year, 7th Semester we are required to make a Minor Project, PowerPoint presentation and submit a Technical Report to the CSE Department. We have done the minor project on “WEB TECHNOLOGY “.

In this project we have done a gaming portal where a person can register him or herself and play offline flash game that is created using JavaScript, html, JSP and CSS. We have included pictures, videos of our site in this given project.

By doing this minor project we enhance our knowledge on given topic. It has also helped us discover our capabilities. Through this project we have come to know about importance of team work.

ABSTRACT

More than half a billion online gamers play games on the internet all over the world with average time spent on online games reaching 3 hours a day. These websites have visitors from all over the world and ranging all age groups. So the popularity of online games cannot be denied. So we have decided to build a gaming site where users will be able to play various different games. Users will be needed to create an account through which they will be able to play the games. The front end of the project like the page design will be done using HTML & CSS. The back end which includes creating the database (Oracle Derby) to maintain the user accounts as well as maintaining and updating their scores will be done using JSP. Finally, the main entity of the project i.e. the games will be created using HTML, JavaScript, CSS. Users can easily log in to their accounts and the games will be present on their home pages which they can easily play by clicking on the respective game logo. The user home page will also include their own data like their highest scores in each game as well as global highest scores in the respective games. Some of the games which will be included in our website are – Crystal Breakout, 4x4 Tic-Tac-Toe, Snake Xenzia, The Maze, Rescue Out Pikachu, Copter Crash, Jumping Kangaroo.

ACKNOWLEDGEMENT

We wish to take this opportunity to convey my sincere thanks to all who in various ways have contributed in the success of this project. Prior to all we would like to thank Mrs. Madhusmita Mishra for initiating us on to this project. Her constant source of guidance, support & inspiration, perspective insight, friendly cooperation, valuable suggestion & advice, which not only helped to complete this project in such a nice way but will also act as stepping stone for my professional career. It has been a great pleasure learning and working under him. We would also like to thank our HOD Mr. Amrut Ranjan Jena for giving us such an opportunity to present out ideas and skills. We would also like to thank our parents for their constant source of inspiration and support

We would also like to thank all the staff member of Computer science department (C.S.E) for their co-operation and constant motivation.

Thanks,

Samibrata Ghosh (131230110092)

Tuhin Dutta (131230110120)

Dipanjan Banerjee (131230110039)

Bastab Dey (131230110030)

Sayak Chatterjee (131230110098)

Shubhadeep Chowdhury (131230110102)

CHAPTER 1

INTRODUCTION

Games are fun to play and enjoyable to program too. Games provide an interesting way of learning all the aspects of a new programming language. So while learning JavaScript, HTML, CSS as a part of our regular course we decided to make a flash gaming portal. So what we have developed is a flash game portal for which we have developed a few interesting games. In order to play these games the user will have to create an account. This account will help the users to play the games of one's choice. The account will also enable the user to view one's highest score in a session and also their top scores in each game till date. To maintain the competitive edge we have included the provisions for users to see the top 5 highest score in each game made by other users. The games have been designed to run on web browsers and have been designed using JavaScript, HTML, and CSS. For the backend of the project i.e. to maintain the database we have used apache derby network server.

The next part of the report will give introduction, design analysis and screenshot of the five games developed. Each chapter will explain one game. The last chapter analyses the database design and DFDs.

CHAPTER 2

4x4 TIC-TAC-TOE

4x4 Tic-Tac-Toe is artificial intelligence based flash game built mainly for children. This is an updated version over the classic 3x3 tic-tac-toe. Since in this version the board is bigger so every next move needs the player to think and plan deeper.

The game consists of two players *X* and *O*; one is the player while the other one is computer(AI). They both take alternating turns marking the spaces in a 4×4 grid. The symbol given to the player is generated randomly every new game(X or O). The player or computer who succeeds in placing four of their marks in a horizontal, vertical, or diagonal row wins the game. If none of the players can place all four of their marks in a single row or column or diagonal but the board is filled then the game is declared draw. The game has three(3) levels namely: -**'Beginner', 'Novice'** and**'Expert'** signify their level of difficulty.

The interface consists of a message bar which displays game status and next player's turn. It also consist of a scoreboard showing **no of games played**, **no of games player won** and **no of games computer won.** The difficulty level is shown separately in yellow rectangle box.

The interface also has three buttons one is **'Start Over'** to play a new game/restart, another is **'Level Up'** to change game difficulty level and the last one is **'Quit'**.

After any valid or invalid move or game status changed to won, lose or draw various informative and exciting sounds are played in background that keeps the game lively.

2.1 INTERFACE DESIGN

The whole interface is designed using HTML and CSS.

2.1.1 Board Design

The 4x4 squares are designed individually using html table tags. All these squares belong to a bigger square holding all 16. All these squares are given onclick listener enabling them to fire an event whenever the user clicks the corresponding square. The smaller 16 squares belongs to the css class ‘smallsq’ and the bigger square belongs to the css class ‘bigsq’ which are designed in the internal style tag.

2.1.2 Message Bar Design

The message bar belongs to css class ‘msgs’ having id ‘message’. This message bar is updated after every event showing whose turn it is or if the user tried an invalid move or if the game is over showing whether player or AI has won.

2.1.3 Level Bar Design

The level bar belongs to css class ‘levelmsgs’ having id ‘levelname’. The level bar is updated every time the player clicks the ‘Level Up’ button. If the current level is ‘Beginner’ it will be updated to ‘Novice’ or if the current level is ‘Novice’ it will be updated to ‘Expert’ and the cycle continues. The game is also restarted.

2.1.4 Scoreboard Design

The scoreboard belongs to div class ‘scorecard’ within which there isthree more div classes in order: ‘scorestat1’, ‘scorestat2’ and ‘scorestat3’. The first class shows the no of games played for the current session and level. The second class shows the no of games player won for the current session and level. The third class shows the no of games computer won for the current session and level. All these classes are designed in the internal style tag. After every game the scoreboard is refreshed. Initially all fields are set to zero.

2.1.5 Button Design

There are three buttons ‘StartOver’, ‘Level Up’ and ‘Quit’. ‘StartOver’ button is designed with div id ‘reset’, ‘Level Up’ button is designed with div id ‘level’ and ‘Quit’ button is

designed with div class ‘backbtn’. All these buttons switches color on focus using hover property.

2.2 AI DESIGN

The AI is mainly designed using base algorithm as minimax and alpha-beta pruning as a patch over it to improve its performance. Apart from these many optimization techniques are used to improve algorithmic complexity and faster response for the AI player.

2.2.1 Minimax Design

A **minimax algorithm** is a recursive algorithm for choosing the next move in an n-player game, usually a two-player game. A value is associated with each position or state of the game. This value is computed by means of a position evaluation function and it indicates how good it would be for a player to reach that position. The player then makes the move that maximizes the minimum value of the position resulting from the opponent's possible following moves. If it is **A**'s turn to move, **A** gives a value to each of his legal moves.A possible allocation method consists in assigning a certain win for **A** as +1 and for **B** as −1. The algorithm can be thought of as exploring the nodes of a game tree.
The effective branching factor of the tree is the average number of children of each node (i.e., the average number of legal moves in a position).The performance of the naïve minimax algorithm may be improved dramatically, without affecting the result, by the use of alpha-beta pruning.

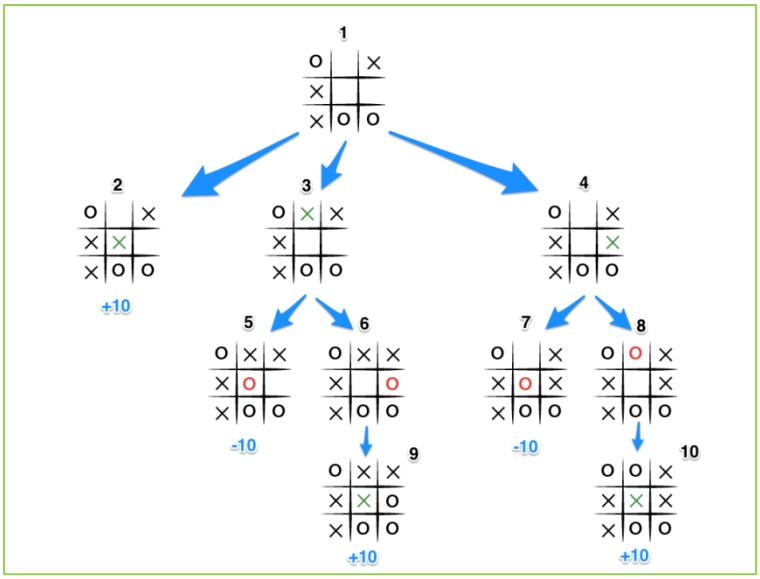


Figure 2.1: A board showing move possibilities and scores in minimax technique.

2.2.2 Alpha-Beta Pruning

Alpha-beta pruning is a way of finding the optimal minimax solution while avoiding searching subtrees of moves which won't be selected. In the search tree for a two-player game, there are two kinds of nodes, nodes representing your moves and nodes representing your opponent's moves.Alpha-beta pruning gets its name from two bounds that are passed along during the calculation, which restrict the set of possible solutions based on the portion of the search tree that has already been seen.Specifically,

β Beta is the minimum upper bound of possible solutions

α Alpha is the maximum lower bound of possible solutions

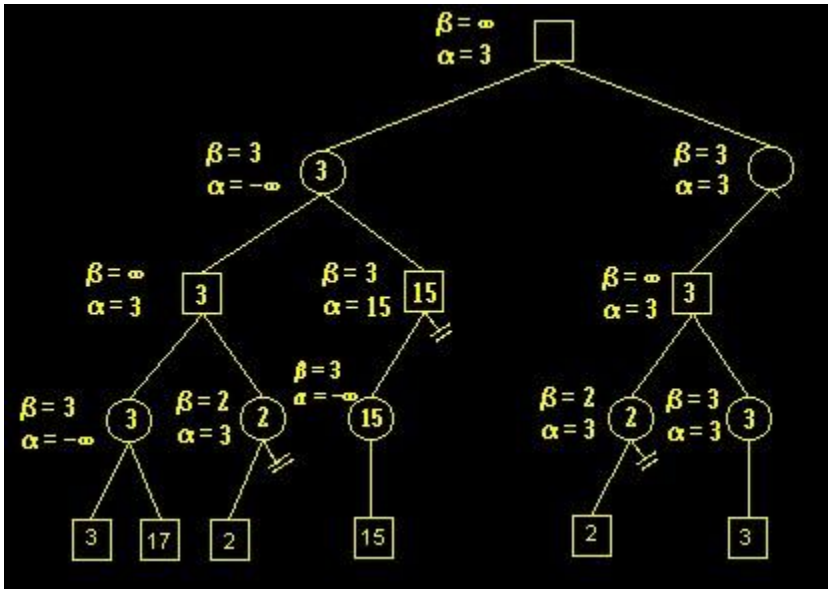


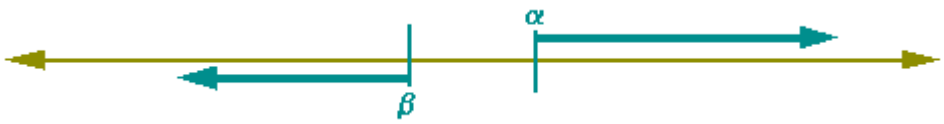
Figure 2.2: A game tree showing irrelevant subtrees

are prunedusing alpha-beta pruning technique.

Thus, when any new node is being considered as a possible path to the solution, it can only work if:

$$\alpha <= N <= \beta$$

where N is the current estimate of the value of the node.



At this point, we know that this node could never result in a solution path that we will consider, so we may stop processing this node. In other words, we stop generating its children and move back to its parent node. For the value of this node, we should pass to the parent the value we changed which exceeded the other bound. The minimax and alpha-beta pruning is used in function called **getBestMove**;

2.2.3 Modules

The various important modules that has been used in developing the AI and other useful utility functions are stated one by one.

2.2.3.1 startGame

This is the function where all the game variables are initialized including setting up of various sounds. It also refreshes the scoreboard table. It also erases away all temporary game data.

At last it sets the initial message in the message bar calls function **playSound.**

2.2.3.2 playSound

This function contains a switch-case structure to determine the type of sound and play it accordingly.

2.2.3.3 compMove

This function is called when its computer's turn to place a move. The function first checks if the game is already over by verifying if the global variables **'winner'** and **'tie'** are set to true otherwise it checks the difficulty level and places its move according to that.

If the difficulty level is of 'Beginner' then the computer moves into any random empty square otherwise if the difficulty level is of 'Novice' or 'Expert' then the computer applies various specific algorithms to find the next optimal move.

2.2.3.4 can_I_win

This function checks if the computer has three in any row. The function finds the empty place in that row where on placing the computer can win. AI then places that winning move on board.

2.2.3.5 isRivalWinning

This function checks if the opponent player has three in any row. The function finds that empty place in that row where on placing the opponent winning position can be blocked. AI then places the blocking move on board.

2.2.3.6 can_I_Fork

This function checks for all board configurations that if a fork move is possible (creating a opportunity to win in two ways).

How does it work?

The function checks if there is a row and a column having a common square such that no opponent symbol is present in either of them. If such kind of a pair is found the AI makes its next move in any of the squares other than the common intersecting one among those available in that row or column provided the row or column contains marks less than 2. Now if both row and column has two marks each then the next move must be in the common intersecting square to create fork.

2.2.3.7 has_user_Forked

This function checks for all board configurations that whether the opponent is trying to create a fork. By looking ahead the AI player decides whether the opponent can create a fork in future. If so AI player places a move in the common intersecting square to destroy opponent's plan.

How does it work?

The function checks if there is a row and a column having a empty common square and both the row and column has no AI's mark. The AI understands that he could be in danger if the opponent succeeds in creating a fork. Thus AI player blocks the intersecting square. Now to keep flexibility in the game the fork is blocked only when opponent has 2 marks in each row and column.

2.2.3.8 Score Calculator

This function is used to calculate the score when the user presses the 'Quit' button and update the database if the score got improved from the previous best. The score calculation strongly depends on the level in which the player is playing. The thresholds that are multiplied have values depending on the level. It's quite easy to win a game in 'Beginner''s level but score increment is slow linear whereas if the level is 'Novice' or

‘Expert’ then winning is quite difficult especially in ‘Expert’ but score increases exponentially after every win. A balance is maintained in computing final score that generalizes the levels, up to a certain point.

2.2.3.9 Other Utility Methods

- **Move_oppCorner :** This function helps in moving to opposite corner if the player has moved to a certain corner.
- **Switch_turn:** This function is called after every move to switch turns.
- **findCommonMove**: This function is used to find the common intersecting square given a set of row and column.
- **nextMove:** This function is used to take input from the player and then calls **Switch_turn**.
- **Chech_for_tie** and **check_for_winner**: These functions are used to check if the game has a result and sets the global variables to true. It also plays the appropriate sound using **playSound** method.

2.3 SCREENSHOT

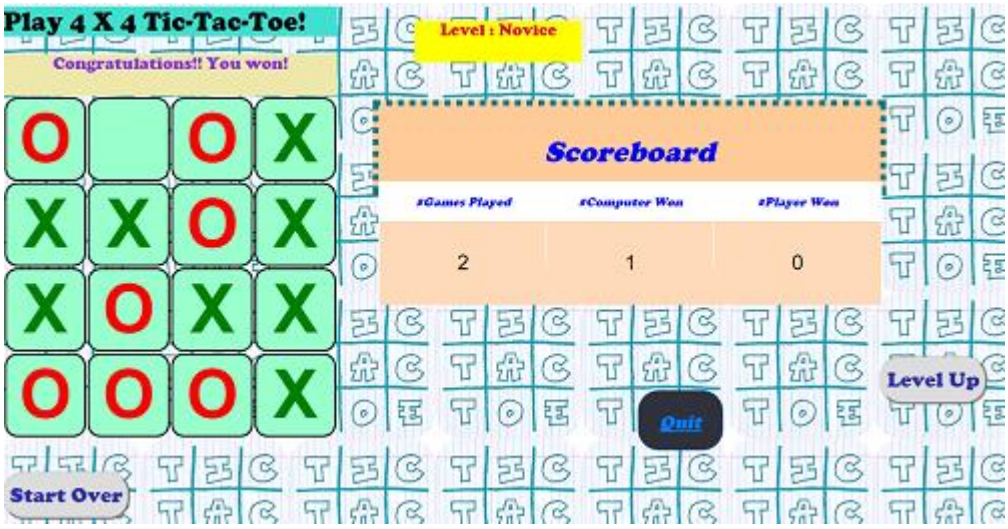


Figure 2.3: A screenshot showing a gameplay of "4x4 Tic-Tac-Toe".

CHAPTER 3

RESCUE OUT PIKACHU

This is a single player game.The objective of the player is to avoid the falling rocks from crashing into Pikachu. (whom the player has to save). The player can control Pikachu using the A and D buttons on the keyboard (A = left and D = right). After every rock has been successfully dodged by pika the score is increased by 100. A player gets 5 lives every round on exhaustment of which the final score is calculated and compared with previous best for the current session if any and updated. Anytime the player can press the quit button after which the highest score in that session is sent to the database and redirected to player's homepage.

3.1 INTERFACE DESIGN

The whole interface is designed using HTML and CSS.

3.1.1 CANVAS DESIGN

The canvas dimensions are width 800px and height 482px with a yellow border. The canvas consists of a landslide background, five round rocks of different color and sizes. Pikachu, a image, whom the player has to save from falling rocks.

Pikachu is a image of dimension 54 x 50 whereas the rocks are all drawn as circles having different sizes.

3.2 GAME DESIGN

I start off by defining the rock objects. The rocks are simple canvas drawn circles with different colors (namely yellow, orange, green, yellow) of radius 50 pixels. Then I define the player i.e. the one that the user will have to save from crashing against the rocks. For displaying the player I have simply made a player object and defined the object as an image by providing the image url.

Then we need to define the initial positions of the player as well as the rocks. I have implemented these functionalities using the init_obj() method.

3.2.1 Init_obj

This function simply initializes the starting position of the player and the rocks.
To add to the difficulty of the game I have made provisions to shuffle the positions of the rocks after each round. This has been implemented in the init() method.

3.2.2 Init

Firstly we clear the canvas after each round, shuffle the positions of the rocks and then repaint the player in the position it was in the previous round. To illustrate the shuffling I will be citing an example:

1. The yellow rock's original position is 50.
2. For the next frame
    a. If yellow rock's position is 50 it becomes 100.
    b. Else it becomes 50.

    Steps 1 and 2 continue until the rocks crash into the player.

Similar things happen for the other rocks too. Once the position of the rocks are determined the rocks are drawn and then the player is redrawn.
For drawing the rocks in the above function I have implemented the drawrocks() function.

3.2.3 Drawrocks

This function simply draws rocks using rock objects defined earlier and x, y parameters set in init() functions. This function draws the rocks by passing on the rock object attributes to the draw1() function defined below.

3.2.4 Draw1

This function uses the context.arc method which takes its parameter inputs from the rock object attributes. The other attributes such as line width strokestyle have also been defined here.
After the function to draw the rocks have been implemented we have to implement the function to draw the player which I have done in the drawImage() function.

3.2.5 drawImage

This is a simple function that draws the player canvas using the attributes of the player object attributes and also sets the initial x, y coordinates of the player.

After all the elements of the game have been defined its time to implement the main aspect of the game i.e. collision

3.2.6 Collision

The procedure used for the collision detection is as follows:

1. If player.x – rock.x <= 60 or abs(player.x – (rock.x – rock.radius)) <= 55 go to step 2.
2. If player.y -50 < rock.y collision is detected.
3. Timescollided increases by 1.
4. The user is alerted that he/she has collided
5. If timescollided is greater than 5 the game ends as the user has 5 lives.

This procedure remains same for all the four rocks. Timescollided is a variable that is required to keep track of lives of the user.

After the collision detection has been done its time to make the player move.

3.2.7 Move

This function simply takes the code of the key pressed by the user as inputs and executes the switch case accordingly. If the key pressed is A then it moves the player left by decreasing the x position by the image size and vice versa when D is pressed. To get the keycode of the key pressed by the user we need to add a keydown listener.

With the elements drawn and collision detection done and players movement completed its time to animate the elements for better feel of the game. This has been implemented in the following functions.

3.2.8 requestAnimFrame

This function is responsible for the continuous falling of rocks at different pace after every frame change.

3.2.9 animate

The following three functions control the whole animation of the game. Each animation has an associated sound with it making the

game more lively.

3.2.9.1 animate_roundRocks

This function executes only when hasCollided variable is set to false. The function first clears the rock object in whose reference it is called. Then its checks that after transition whether the rock is getting out of the canvas. If so then the rock is resettled to original position else the redrawn after transition and this process goes on. A pebbles and stone falling sound is played in background.

3.2.9.2 animate_pikaLeft

This function is called whenever ‘A’ is pressed from keyboard. The function at first erases the current pika image then redraws it after transition. Every transition results in shifting pika to the left by 25px. Pikachu shouts ‘pika pika’ whenever this function is called.

3.2.9.3 animate_pikaRight

This function is called whenever ‘D’ is pressed from keyboard. The function at first erases the current pika image then redraws it after transition. Every transition results in shifting pika to the right by 25px. Pikachu shouts ‘pika pika’ whenever this function is called.

3.3 SCREENSHOT

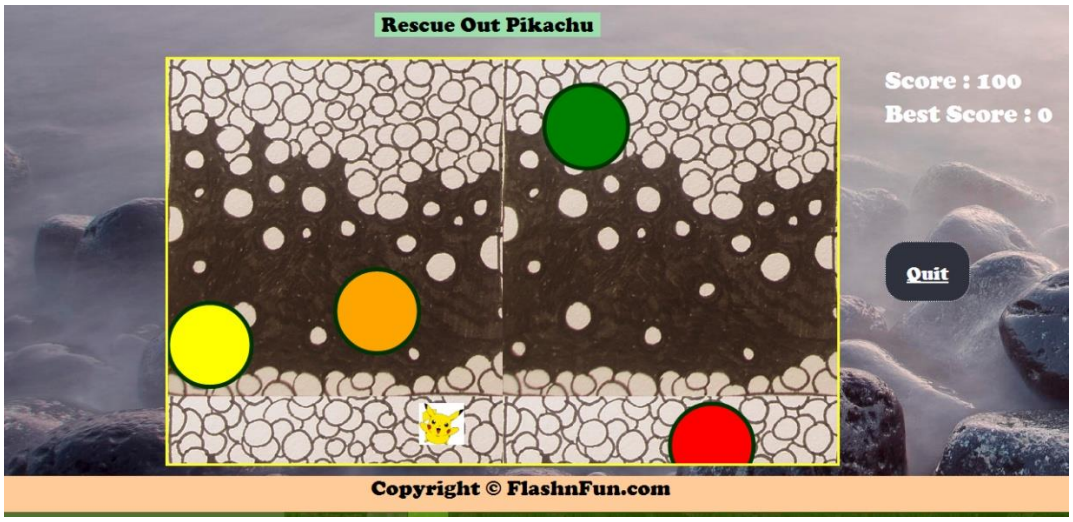


Figure 3.3: When the player wins the match in the game RESCUE OUT PIKACHU

CHAPTER 4

CRYSTAL BREAKOUT

Crystal Breakout is a simple 2d arcade flash game built mainly for the children. It is quite similar to that of the Atari breakout but we have given it a newer look and design.

In this game we will get to see a layer of crystals lines the top of the canvas. A ball travels across the canvas bouncing off the top and side walls of the canvas and a paddle to hit the ball. When the crystals are hit, the ball bounces away and the crystal is destroyed. The player will lose a turn or life if the ball touches the ground of the canvas. To prevent this from happening the player has a movable paddle to bounce the ball upward, keeping it in play. The game has only 1 level. The player will get to see his score and his best score on the screen itself. The interface has two button one is **"Play"** button and the other one is **"Quit"** button.

The game is provided with exciting sounds that are played in background to make the game lively.

4.1 INTERFACE DESIGN

The whole interface is designed using HTML and CSS.

4.1.1 Create the Canvas

The Html document structure is quite simple, as the game is totally rendered on **<canvas>** element.

The **<canvas>**element has an id of **"mycanvas"** to allow us to easily grab a reference to it and it is 750pixel wide and 500pixel high.

```
var canvas = document.getElementById("myCanvas");
var ctx = canvas.getContext("2d");
```

Here we are storing the reference to the **<canvas>**element to the canvas variable. Then we are creating a **"ctx"** variable to store the 2D rendering variable which is the actual tool we use to paint on the canvas.

4.1.2 Button Design

There are two buttons **'Play'** and **'Quit'**. 'Play' button is designed with id **'play'** and on clicking the play button your game will start, 'Quit' button is designed with div class **'back'**and on clicking this quit button it will take the user to his homepage. All these buttons are designed with cascading style sheet.

4.1.3 Crystal and Paddle Design

```
ctx.beginPath();
ctx.rect(brickX, brickY, brickWidth, brickHeight);
 ctx.fillStyle = "#55d4af";
ctx.fill();
ctx.closePath();
```

To design the crystals of our game, we have written all the instruction within **beginPath()** and **closePath()** methods. As our crystals are rectangular in shape that's why we have used **rect()** method. The first two values specify the top left corner of our canvas, while the second two specifies the width and height of the crystal. The fillStyle property stores the color of the crystals that will be used by the **fill()** method.

4.1.4 Ball Design

```
ctx.beginPath();
ctx.arc(x, y, 10, 0, Math.PI*2);
ctx.fillStyle = "#FF002A";
ctx.fill();
ctx.closePath();
```

As we all know a ball is circle in shape that's why we have used **arc()** method. Where x and y the co-ordinates of arc center and arc radius.

4.2 GAMEPLAY

Crystal breakout begins with eight rows and seven columns of same color and of same point i.e. 50. The player has to knock out as many crystals as possible by the help to the wall and paddle. If the player's paddle misses the ball's rebound, then the player will lose a turn. The player will be given three lives and on every one miss he will lose one life. The ball speed will be increased on every loss of life and if the player able knock out all the crystals then his score will be 2800 which is the maximum score in this game.

4.2.1 Collision Detection

We have talked about the gameplay. Now, in this portion will see how the collision detection is done and how the crystals disappear when it is hit. We have to check that if the center of the ball is inside the co-ordinates of one our brick then we'll change the direction of the ball. For this the following statement needs to be true:

- X position of the ball is greater than the x position the crystal.
- X position of the ball is less than the x position of the crystal plus its width.
- Y position of the ball is greater than the y position of the crystal
- Y position of the ball is less than the y position of the brick plus its height.

Then we have initialized status property to each crystal object, if the status is 1 then the crystal will be drawn and if it is 0 then the crystal is hit by the ball and it disappears from the canvas.

4.2.2 Game Over

The logic behind losing in this game is very simple. If the player misses the ball with the paddle and the ball hit the bottom part of the canvas, then the player will lose one life at a time. If the player losses three lives then the game is over and this function will be called **youLose()**which display a message **"YOU LOSE"** in red color.

Similarly, when the player completes the game without losing a single life then also the game will be over and this time different function will be called i.e. **yonWon()**which display a message **"YOU WIN"** in white color.

Figure 4.1: When the player loses the game

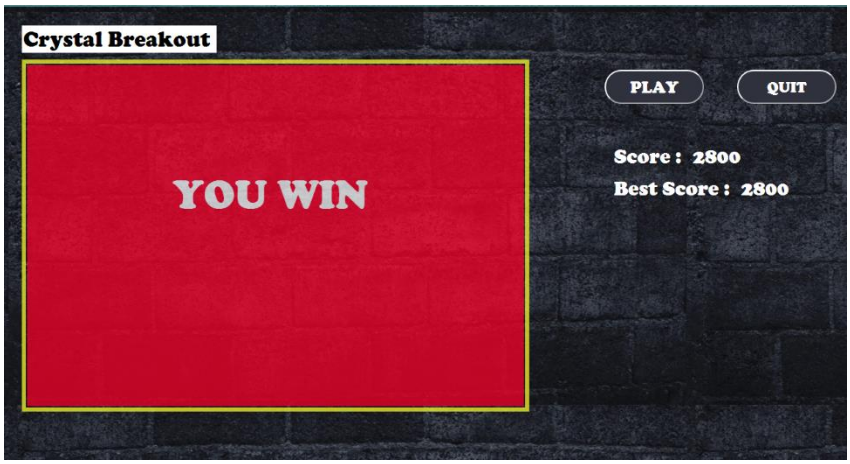


Figure 4.2: When the player wins the match

4.2.3 Track your Score

In this game the player will be able to track is score on the canvas as well as on the screen itself.

The player will be awarded points each time he hits the crystal. The player can achieve a maximum point of 2800.

4.3 SNAPSHOT

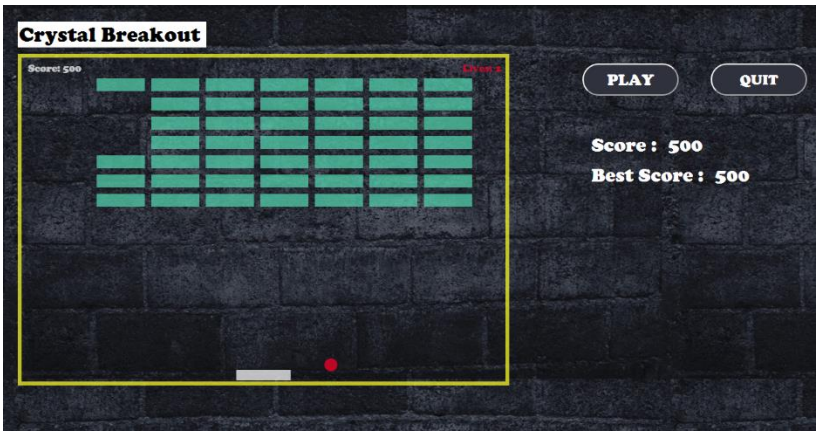


Figure 4.3: A snapshot of the game “Crystal Breakout”.

CHAPTER 5

SNAKE CLASSICS

5.1 ABOUT THE GAME

Snake Classics is a game where the player maneuvers a block which grows in length, with the block itself being the primary obstacle. This game is the customized and modified version of the game that we used to play in the old nokia phones in the 1990s. The player controls a small block, resembling a snake, which roams around on a bordered canvas, picking up food, trying to avoid hitting its own tail or the walls that surrounds the playing area. This game works in a FIFO concept where the data is stored in a queue. Each time the snake eats a piece of food, the player should try to move the snake in a direction, avoiding the snake to hit its own tail or the walls that surround the playing area. Each time the snake eats a piece of food, its tail grows longer, making the game increasingly difficult. As the snake keeps on growing and the player scores points, after scoring some points, the color of the snake changes at frequent intervals resembling that the player has reached a level harder than the previous one. It becomes difficult to control the snake as it grows longer. The user controls the direction of the snake's head (w=up, s=down, a=left, d=right) and the snake's body follows. The player cannot stop the snake from moving while the game is in progress and cannot make the snake go in reverse. There is as **start button** where the player clicks and moves to the second page where the game is to be played. Inside a green canvas the playing arena is created where the snake snatches it food and if it fails the game gets over and the score is displayed in the screen and the player is redirected to the first page.

Some of the interactive sounds are provided in the background to create the atmosphere of excitement when the gameplay is in progress.

5.2 INTERFACE DESIGN

The whole interface is designed using HTML, JavaScript and CSS.

5.2.1 Grid Design

The game grid is made up of cells where the game is confined in absolute sized chunks of data. Each time the snake moves the value of the grid gets changed. The values of the grid increases or decreases according to the control of the user. The width and height denotes the amount of columns and rows of the grid respectively. The init function (discussed later) initializes a c*r grid with the direction variable d, where the variables c and r represents the updated width and height of the grid respectively. The set and get variables are used for setting and getting the values of the grid respectively as shown in the figure below.

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	1	1	1	1	0
0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	0
0	2	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Figure 5.1: A table showing the grid where the snake moves.

5.2.2 Canvas Design

The canvas is used for graphical representation of the game entitled SNAKE CLASSICS. It uses mainly two types of fill functions namely- fillStyle and fillRect. The fillStyle function is used for coloring the snake and food. It is also used to change the text color. The fillRect function draws a rectangle at the canvas at the(x,y) co-ordinate. The context of the canvas is two dimensional in nature. I have also added the shadowcolor and shadowblur functions inside the canvas to increase the look and feel of the game. A green border is also added outside the canvas as it represents that the snake is moving in a grass.

5.2.3 Button Design

There are two buttons in the game namely- Start and Quit. The start button is placed in the first page before the game starts and the Quit button is placed in the gaming page. Both the buttons are designed using CSS. Box shadowing is also done in the start button

and shadow color is also included as if when the player clicks the button it changes color. The quit button quits the game and returns in the homepage of our website.

5.2.4 Methods

The various important modules that has been used in developing the AI and other useful utility functions are stated one by one.

5.2.4.1 windows.onload

All the functions are embedded inside onload function. This function produces background sounds and all the working principles of buttons are embedded inside this function. This function also displays the score in the screen after the game gets over.

5.2.4.2 setFood

All the essential functions of the food item are performed using setFood() function. The food item is represented by an id in the grid. The position of the food is set randomly to a free cell in the grid. The setFood() function sets a food id at a random free location. The food is set in any position in the x and y-coordinates respectively. Once a snake snatches the food the direction and position of the food gets changed and it is placed in any random free location.

5.2.4.3 init

The most important part of the game , i.e, the grid is set and initialised by using the init() method. It determines the direction and position at which the grid is set. It initializes all the basic parts of the game viz, snake color, score and the directions to which the food(involuntary) and the snake(voluntary) will move. The stack pointer of the x and y coordinates is also set using the init method. This method is also called after the update() method to set the positions of the snake and the grid after increasing score.

5.2.4.4 draw

The major properties of fillStyle and fillRect functions are embedded in the draw() function. It is mainly used to draw the snake and the food and also set its colors using switch case. The score is also set using this function. The fillRect functions are used to draw the snake and its food and the fillStyle functions are used to color both and also gives a detailed design of the score.

5.2.4.5 update

The update() method is also one of the vital methods designed in the game. It is used for updating the scores and positions of the food. This method is used when the tail of the

snake increases in length after the snake snatches its food. This method also helps in the movement of the snake controlled by the up, down, left and right keys(w,s,a,d) respectively. When at the beginning, the snake does not snatch the food and touches the wall, the snake in its initial position comes from a random direction. This method also increases the scores by 10 points and sets the diescore on the screen if the snake touches itself after getting longer or the walls.

5.2.4.6 loop

The loop() method is the animated interface method in the complete game. Using the method window.requestAnimationFrame() animation is performed in the game. This function is mainly used to change the color of the snake at certain bounds of increasing scores indicating the raise in level of the game. A sound is played after the snake changes its color. When the snake dies, the score is displayed at the center of the screen indicating the termination of the game. It also stops the theme music after the snake dies. This function also redraws the canvas after the termination of the game.

5.2.4.7 endGame

The endGame() function resets the game and redirects to the first page where the start button is displayed indicating the user to play the game again.

5.3 SNAPSHOT:

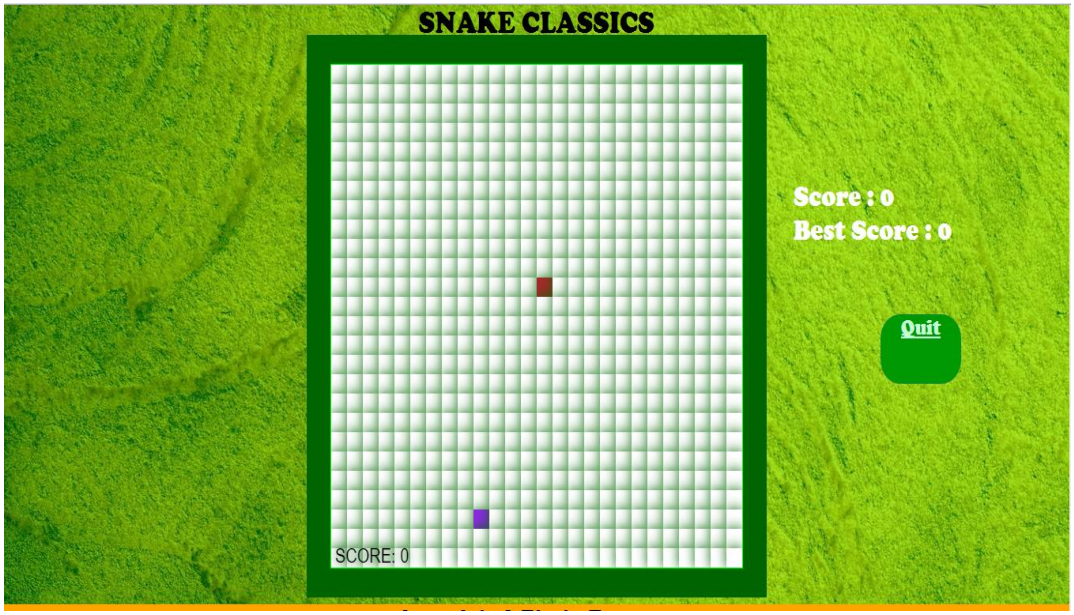


Figure 5.2: A screenshot showing a gameplay of the game "**SNAKE CLASSICS"**.

CHAPTER 6

THE MAZE GAME

6.1 ABOUT THE GAME

- The game is a single player game.
- The game generates a different maze each time it is started.
- The player has to solve the maze i.e. reach the yellow tile at the rightmost bottom of the maze.
- When the player reaches the yellow tile he/she gets a message that the maze has been solved and he/she gets to know ones score.
- After knowing the score the maze gets reset and the player can start playing again.
- The player can quit the game anytime by pressing the quit button by clicking on the quit button on the page.
- When the player quits the game he/she is able to see their highest score in that particular session as the best score.

6.2 GAME DESIGN

The main functions and algorithms that have been deployed to design the game have been described below:

I have used the recursive backtracking algorithm to design the maze which is one of the basic algorithms to design a maze. The recursive backtracking algorithm can be described as:

1. Choose a starting point in the field.
2. Randomly choose a wall at that point and carve a passage through to the adjacent cell, but only if the adjacent cell has not been visited yet. This becomes the new current cell.

3. If all adjacent cells have been visited, back up to the last cell that has uncarved walls and repeat.
4. The algorithm ends when the process has backed all the way up to the starting point.

I have implemented the algorithm in JavaScript using the following functions:

6.2.1 Mazegenerate

This function initializes an array named generate with all 0s. The array contains only 0s and 1s where 1 represent path and 0 represent walls. So initially the maze is composed of only walls. By default I choose (0,0) index of the generate array as the starting point. After this the control gets transferred the to the recursive function carvepath() which has been described below. The function returns the final state of the array generate which is used to draw the maze.

6.2.2 Carvepath

This is the recursive function that makes out the path in the maze according to the following procedure:

1. Generate an integer array with 4 random numbers to represent directions.
2. Start a for loop to go for 4 times.
3. Set up a switch statement to take care of 4 directions.
4. For that direction, check if the new cell will be out of maze or if it's a path already open. If so, do nothing.
5. If the cell in that direction is a wall, set that cell to path and call recursive method passing the new current row and column.
6. Done.

To produce an array with 4 random numbers representing directions I have used the function array.prototype.shuffle() which has been described below.

6.2.3 array.prototype.shuffle

An array named direction has been declared which contains 4 integers (1, 2, 3, 4) which represents the four directions namely up, down, left, right. This function array with the array elements shuffled. The procedure to shuffle the array is as follows:

1. Start at the end of our array by selecting the last item.
2. Then using the random function() select a random number from 0 to (size of the array - 2).

3. Then just swap the last element of the array with the element that is present in the array at the randomly selected index.
4. Repeat steps 1, 2, 3 for the second last element of the array and so on till the first element of the array.

Successful implementation of these function gives us the generate array which helps in designing the maze. Now we need to draw the maze which I have implemented in the draw function.

6.2.4 Draw

For drawing on the webpage I have used HTML canvas. The maze that I have drawn has red and black tiles. The red tiles represent path and the black tiles represent wall. The generate array is used to draw the maze in the following manner:

1. Whenever the element in the array is 1 a red tile is drawn.
2. Whenever the element in the array is 0 a black tile is drawn.

Successful completion of all the iterations draws the complete maze on the webpage. After the maze has been drawn we need to make a player which I have done in the myplayer() function.

6.2.5 Myplayer

This function simply draws a 5*5 square at the starting point [i.e.(0,0) of the maze]. The next thing to implement is to make the blue player move using the arrow keys or WASD. The first step is to calculate the new X and Y-coordinate of the blue rectangle. The second step is to move the rectangle, if it can move, and to show a "Congratulations!" message if the player reached the end point (the yellow tile). This functionality has been implemented in the move function

6.2.6 Move

This function takes the keycodes of the keys pressed by the user as an input and uses a switch case to move the player. For example the initial position of the player is represented by its x and y coordinates as x =0 and y=0. If the user presses the up W(up) key the y coordinate of the player decreases by 5(as the player's size is 5*5) and the x coordinate is the same. Correspondingly when down key is pressed the y coordinate is increased by 5 and so on. After the execution of the switch case the player is deleted from the previous position and drawn in the new position using the new x and y coordinates. This procedure enables the player to move about the maze.

In addition to this a keydown listener is needed to get the code of the key pressed by the user.

The above function allows the player to move about the maze. But the player cannot move through the wall. So we need to detect whether the tile in which the player is going to move is a wall or a path i.e. red or black. The collision() function that I have written implements this.

6.2.7 Collision

The logic behind it:

1. Check whether the blue rectangle would move inside the bounds of the canvas
2. If that's the case, get the image data of the rectangle with x = newX, y = newY, width = 5, height = 5. I take 5 as width and height because that's the size of the player. Here newX and newY represent the new position of the played calculated using according to the key pressed by the user.
3. Use a for loop to look at all pixels: if any of them has the color black, then the blue rectangle can't move. If any of them has the color yellow, then you reached the end point.

This function returns whether the player can move or not to the next tile.

Now we need to display the score according to the number of moves the user has used up to reach the ending point. The scoring technique is explained below.

6.2.8 Finalscore

The game starts from leftmost top and ends at the right lowermost corner of the maze. So whenever the user presses the left or up arrow it mostly implies that the user has reached a dead end and is trying to backtrack to get to the correct path. So the user is penalized for every up or left move (only if up or left move is greater than 40). So the final score is displayed as the [number of moves – (total number of up and left move)].

6.3 SNAPSHOT

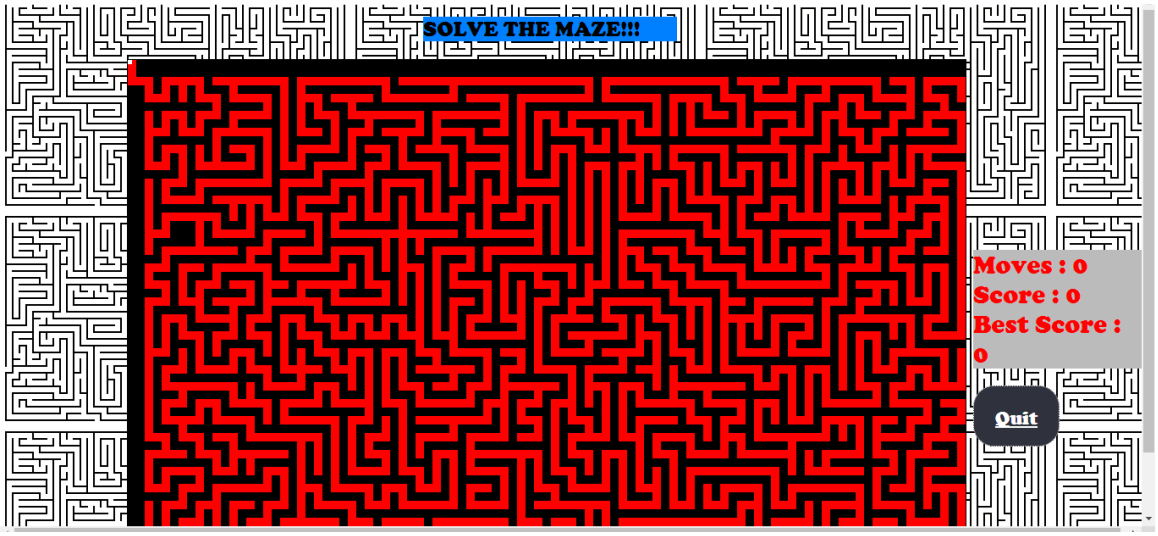


Figure 6.1: A screenshot showing a gameplay of “The Maze Game”.

CHAPTER 7

DATABASE DESIGN

Derby is a full-featured, open source relational database management system (RDBMS) that is based on Java technology and SQL.The client connection URL contains network information (hostname and portnumber) .This information tells the client driver the location of the Network Server. The client driver sends requests to and receives responses from the Network Server.In this activity the Derby database engine is embedded in the Network Server and returns data to the ij client (a client/server configuration).

‘ url = "jdbc:derby://localhost:1527/Flashgames_DB;create=true;user=FlashFun; password=flashfun";

Class.forName("org.apache.derby.jdbc.ClientDriver").

newInstance();

connection = DriverManager.getConnection(url); ’

7.1 SCHEMA DIAGRAM

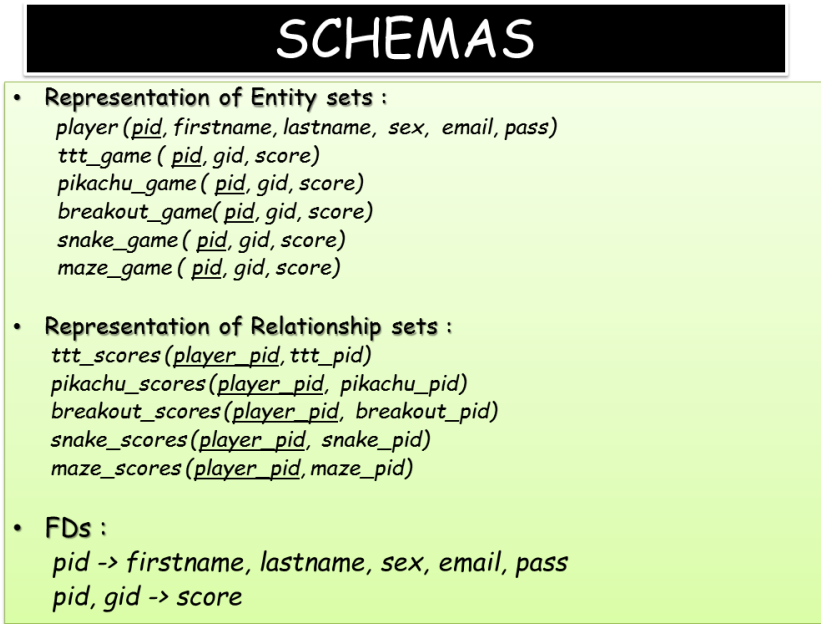


Figure 7.1: A figure showing the database schema

7.2 E-R DIAGRAM

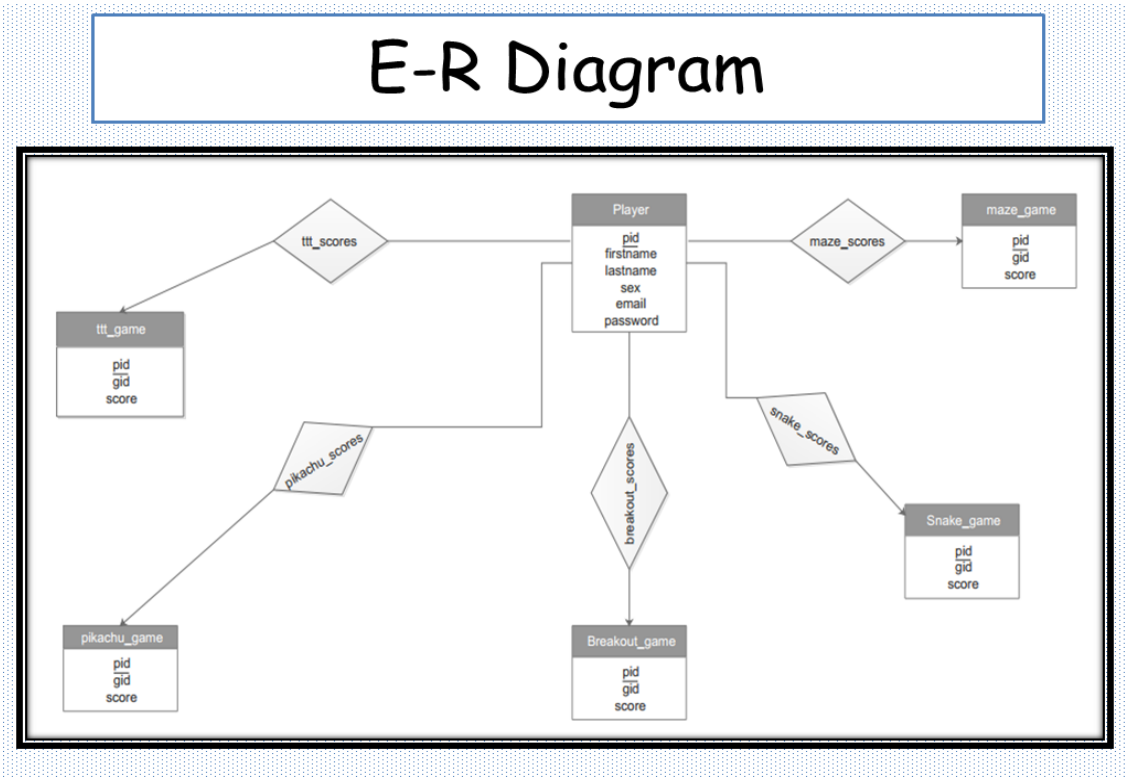


Figure 7.2: A figure showing the E-R diagram

7.3 DATA FLOW DIAGRAMS

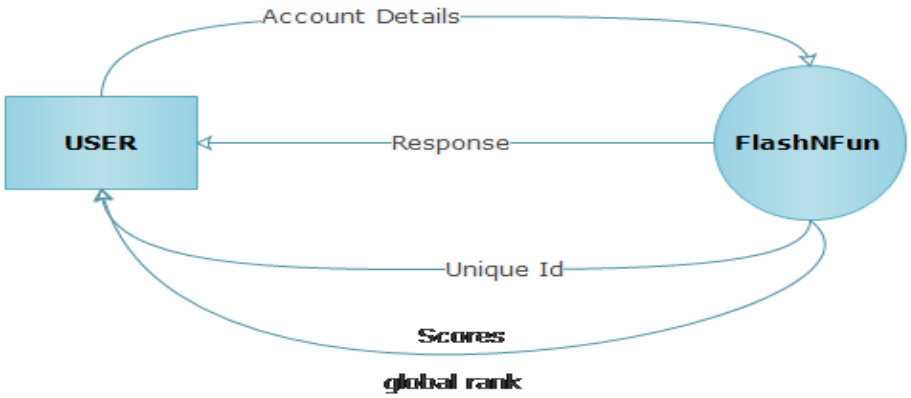


Figure 7.3: A figure showing context level DFD

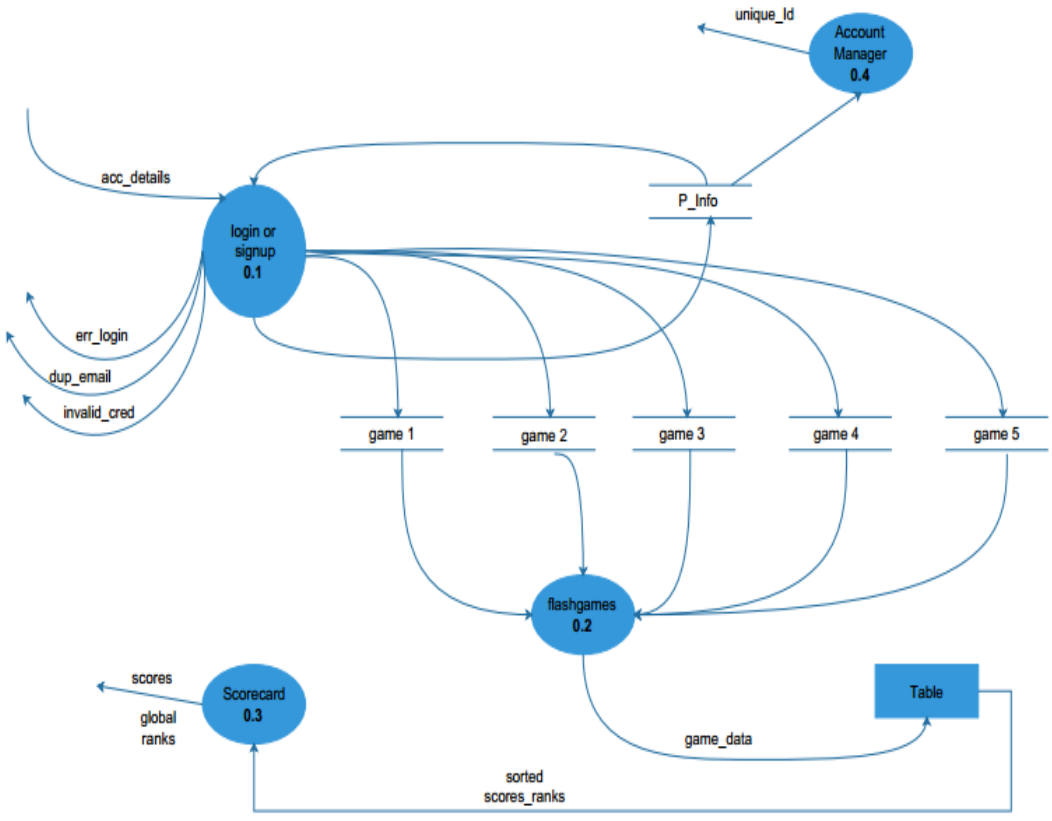


Figure 7.4: A figure showing level 1 DFD

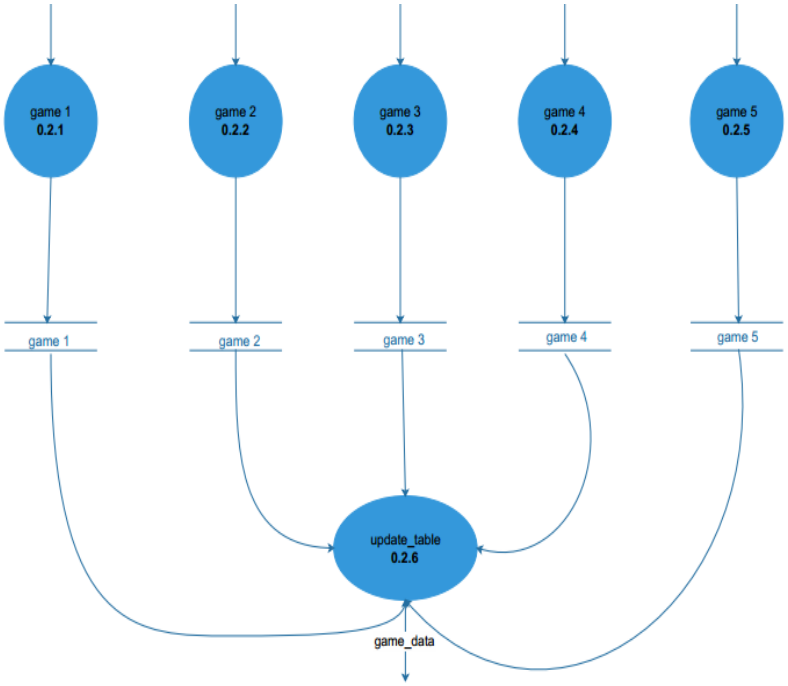


Figure 7.5: A figure showing level 2 DFD

CONCLUSION

As mentioned earlier a lot of people spend hours over the internet playing games. Games are an excellent source of entertainment for the young and old alike. For children and teen it acts as an excellent way of time pass and for the mature population it acts as a very good way of removing stress. Keeping that in mind we would definitely love to keep updating our website in the future. Due to the shortage of time provided for this project we could develop only five games to give a demonstration for our idea. But it's a non-ending process. We will be keep adding new games as new ideas come to our minds to keep our users glued to the portal. Till then we hope you have good time playing the already present games!

REFERENCES

The idea behind the project has been taken from the following books: -

S.L NO	BOOK'S NAME	AUTHOR'S NAME
1	Internet & World Wide Web- How To Program	Paul and Harvey Deitel, Nieto.
2	Web Technologies	Achyut S Godbole, Atul Kahate
3	Web Technology & Design	C Xavier
4	An Introduction to XML and Web Technologies	Anders Moller

Some of our internet references include: -

- **http://www.w3schools.com/**
- **http://www.tutorialspoint.com/**
- **http://www.stackoverflow.com/**