Project Report on

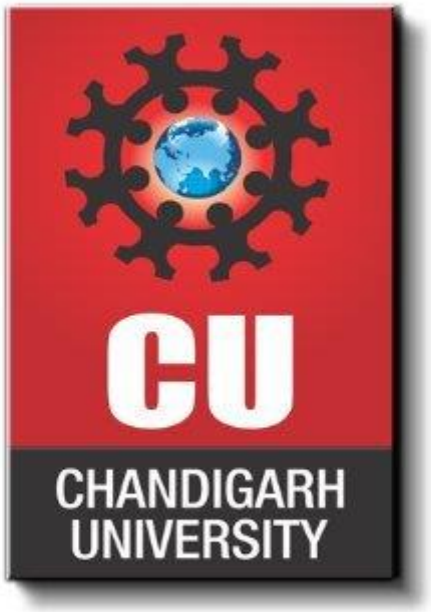
Packman

Submitted By: Abhishek Kumar,

UID – 24MCC20006

Under the Guidance of

Mrs. Winky Bhatia



University Institute of Computing,

Chandigarh University,

Mohali, Punjab

INDEX

Table of Content	
• **Introduction** • **Objective** • **Overview** • **Implementation Details** • **Functions Summary** • **Output Example** • **Advantage and Limitation** • **Conclusion**	

Project Report on Packman

Introduction

Pac-Man is a classic arcade game that was first released in 1980 by Namco. The game has become a cultural icon and is recognized worldwide. The objective of this project is to create a simplified version of the Pac-Man game, allowing players to navigate a maze, collect pellets, and avoid ghosts. This report outlines the project's objectives, implementation details, and the advantages and limitations of the game.

Objectives

The primary objectives of this project are:

To develop a playable version of the Pac-Man game using programming languages and game development frameworks.

To understand the game mechanics, including player movement, collision detection, and enemy AI.

To provide an engaging user experience while maintaining the core elements of the original game.

Overview of the Packman

The Pac-Man game consists of a player-controlled character (Pac-Man) that moves through a maze, collecting dots (pellets) while avoiding four ghosts. The game ends when the player collects all the pellets or loses all their lives. The game features power pellets that allow Pac-Man to eat the ghosts temporarily, turning them blue and making them vulnerable.

Implementation Details

4.1 Development Environment

Programming Language: HTML, CSS, JS

IDE: Visual Studio Code

4.2 Game Components

- Player Class: Represents Pac-Man, handling movement and collision detection.
- Ghost Class: Represents the ghosts, implementing basic AI for movement.
- Maze Class: Defines the layout of the maze, including walls and pellet positions.
- Game Loop: Manages the game state, including rendering graphics and processing user input.

4.3 Key Features

- Player Movement: Controlled using arrow keys.
- Collision Detection: Checks for collisions with walls, pellets, and ghosts.
- Score System: Tracks the player's score based on collected pellets.
- Game Over Condition: Ends the game when all lives are lost or all pellets are collected.

Functions Summary

- move_player(): Updates the player's position based on user input.
- check_collision(): Detects collisions between the player and other game objects.
- update_ghosts(): Updates the position of ghosts based on simple AI logic.
- draw_maze(): Renders the maze and game objects on the screen.
- update_score(): Increases the score when pellets are collected.

Output Example

The game displays a graphical interface with the maze, Pac-Man, ghosts, and score. Below is a textual representation of the game state:

Run

Copy code

Score: 150

Lives: 2

Maze:

##########

#........#

#.#.####.#

#.#......#

#.#.####.#

#........#

##########

Advantages and Limitations

Advantages

- Educational Value: The project provides insights into game development concepts such as game loops, collision detection, and AI.
- Engagement: The game is fun and engaging, appealing to a wide audience.
- Customizability: The code can be easily modified to add new features or improve gameplay.

Limitations

- Simplified AI: The ghost AI is basic and does not replicate the complexity of the original game.
- Graphics: The graphics are simple and may not appeal to players accustomed to modern game visuals.
- Limited Features: The game lacks advanced features such as levels, power-ups, and sound effects.

Code used :

```
<!DOCTYPE html>
<html>
<head>
    <title>Packman By Aditya Rana</title>
    <link rel="stylesheet" type="text/css"
href="https://fonts.googleapis.com/css?family=Permanent+Marker">
    <style type="text/css">
        body {
            background-color: black;
        }

        #pacman {
            height: 470px;
            width: 382px;
            border-radius: 5px;
            margin: 20px auto;
        }

        #shim {
            font-family: 'Permanent Marker', cursive;
            position: absolute;
            visibility: hidden
        }

        h1 {
            font-family: 'Permanent Marker', cursive;
            text-align: center;
            color: yellow;
        }

        body {
            width: 342px;
            margin: 0px auto;
            font-family: sans-serif;
        }
```

```
        p {
            text-decoration: none;
            color: #0000FF;
        }
    </style>
</head>
<body>
    <div id="shim">shim for font face</div>
    <h1>Maintenance</h1>
    <p style="text-align:center;">Aditya Rana</p>
    <div id="pacman"></div>

    <script type="text/javascript"
src="https://cdnjs.cloudflare.com/ajax/libs/modernizr/2.8.3/modernizr.min.js"></script>
    <script type="text/javascript"
src="https://cdnjs.cloudflare.com/ajax/libs/jquery/2.1.3/jquery.min.js"></script>
    <script type="text/javascript">
        /*jslint browser: true, undef: true, eqeqeq: true, nomen: true, white: true */
/*global window: false, document: false */

/*
 * fix looped audio
 * add fruits + levels
 * fix what happens when a ghost is eaten (should go back to base)
 * do proper ghost mechanics (blinky/wimpy etc)
 */

var NONE        = 4,
    UP          = 3,
    LEFT        = 2,
    DOWN        = 1,
    RIGHT       = 11,
    WAITING     = 5,
    PAUSE       = 6,
    PLAYING     = 7,
    COUNTDOWN   = 8,
    EATEN_PAUSE = 9,
    DYING       = 10,
    Pacman      = {};

Pacman.FPS = 30;

Pacman.Ghost = function (game, map, colour) {

    var position  = null,
        direction = null,
        eatable   = null,
        eaten     = null,
        due       = null;

    function getNewCoord(dir, current) {

        var speed  = isVunerable() ? 1 : isHidden() ? 4 : 2,
            xSpeed = (dir === LEFT && -speed || dir === RIGHT && speed || 0),
            ySpeed = (dir === DOWN && speed || dir === UP && -speed || 0);

        return {
            "x": addBounded(current.x, xSpeed),
            "y": addBounded(current.y, ySpeed)
        };
    };
```

```
    ctx.quadraticCurveTo(left, top, left + (s/2),  top);
    ctx.quadraticCurveTo(left + s, top, left+s,  base);

    // Wavy things at the bottom
    ctx.quadraticCurveTo(tl-(inc*1), base+high, tl - (inc * 2),  base);
    ctx.quadraticCurveTo(tl-(inc*3), base+low, tl - (inc * 4),  base);
    ctx.quadraticCurveTo(tl-(inc*5), base+high, tl - (inc * 6),  base);
    ctx.quadraticCurveTo(tl-(inc*7), base+low, tl - (inc * 8),  base);
    ctx.quadraticCurveTo(tl-(inc*9), base+high, tl - (inc * 10), base);

    ctx.closePath();
    ctx.fill();

    ctx.beginPath();
    ctx.fillStyle = "#FFF";
    ctx.arc(left + 6,top + 6, s / 6, 0, 300, false);
    ctx.arc((left + s) - 6,top + 6, s / 6, 0, 300, false);
    ctx.closePath();
    ctx.fill();

    var f = s / 12;
    var off = {};
    off[RIGHT] = [f, 0];
    off[LEFT]  = [-f, 0];
    off[UP]    = [0, -f];
    off[DOWN]  = [0, f];

    ctx.beginPath();
    ctx.fillStyle = "#000";
    ctx.arc(left+6+off[direction][0], top+6+off[direction][1],
            s / 15, 0, 300, false);
    ctx.arc((left+s)-6+off[direction][0], top+6+off[direction][1],
            s / 15, 0, 300, false);
    ctx.closePath();
    ctx.fill();

};

function pane(pos) {

    if (pos.y === 100 && pos.x >= 190 && direction === RIGHT) {
        return {"y": 100, "x": -10};
    }

    if (pos.y === 100 && pos.x <= -10 && direction === LEFT) {
        return position = {"y": 100, "x": 190};
    }

    return false;
};

function move(ctx) {

    var oldPos = position,
        onGrid = onGridSquare(position),
        npos   = null;

    if (due !== direction) {
```

```
keyMap[KEY.ARROW_LEFT]  = LEFT;
keyMap[KEY.ARROW_UP]    = UP;
keyMap[KEY.ARROW_RIGHT] = RIGHT;
keyMap[KEY.ARROW_DOWN]  = DOWN;

function addScore(nScore) {
    score += nScore;
    if (score >= 10000 && score - nScore < 10000) {
        lives += 1;
    }
};

function theScore() {
    return score;
};

function loseLife() {
    lives -= 1;
};

function getLives() {
    return lives;
};

function initUser() {
    score = 0;
    lives = 3;
    newLevel();
}

function newLevel() {
    resetPosition();
    eaten = 0;
};

function resetPosition() {
    position = {"x": 90, "y": 120};
    direction = LEFT;
    due = LEFT;
};

function reset() {
    initUser();
    resetPosition();
};

function keyDown(e) {
    if (typeof keyMap[e.keyCode] !== "undefined") {
        due = keyMap[e.keyCode];
        e.preventDefault();
        e.stopPropagation();
        return false;
    }
    return true;
};

function getNewCoord(dir, current) {
    return {
        "x": current.x + (dir === LEFT && -2 || dir === RIGHT && 2 || 0),
```

```
            "y": current.y + (dir === DOWN && 2 || dir === UP    && -2 || 0)
        };
    };

    function onWholeSquare(x) {
        return x % 10 === 0;
    };

    function pointToCoord(x) {
        return Math.round(x/10);
    };

    function nextSquare(x, dir) {
        var rem = x % 10;
        if (rem === 0) {
            return x;
        } else if (dir === RIGHT || dir === DOWN) {
            return x + (10 - rem);
        } else {
            return x - rem;
        }
    };

    function next(pos, dir) {
        return {
            "y" : pointToCoord(nextSquare(pos.y, dir)),
            "x" : pointToCoord(nextSquare(pos.x, dir)),
        };
    };

    function onGridSquare(pos) {
        return onWholeSquare(pos.y) && onWholeSquare(pos.x);
    };

    function isOnSamePlane(due, dir) {
        return ((due === LEFT || due === RIGHT) &&
                (dir === LEFT || dir === RIGHT)) ||
            ((due === UP || due === DOWN) &&
             (dir === UP || dir === DOWN));
    };

    function move(ctx) {

        var npos        = null,
            nextWhole   = null,
            oldPosition = position,
            block       = null;

        if (due !== direction) {
            npos = getNewCoord(due, position);

            if (isOnSamePlane(due, direction) ||
                (onGridSquare(position) &&
                 map.isFloorSpace(next(npos, due)))) {
                direction = due;
            } else {
                npos = null;
            }
        }

        if (npos === null) {
```

```
            npos = getNewCoord(direction, position);
        }

        if (onGridSquare(position) && map.isWallSpace(next(npos, direction))) {
            direction = NONE;
        }

        if (direction === NONE) {
            return {"new" : position, "old" : position};
        }

        if (npos.y === 100 && npos.x >= 190 && direction === RIGHT) {
            npos = {"y": 100, "x": -10};
        }

        if (npos.y === 100 && npos.x <= -12 && direction === LEFT) {
            npos = {"y": 100, "x": 190};
        }

        position = npos;
        nextWhole = next(position, direction);

        block = map.block(nextWhole);

        if ((isMidSquare(position.y) || isMidSquare(position.x)) &&
            block === Pacman.BISCUIT || block === Pacman.PILL) {

            map.setBlock(nextWhole, Pacman.EMPTY);
            addScore((block === Pacman.BISCUIT) ? 10 : 50);
            eaten += 1;

            if (eaten === 182) {
                game.completedLevel();
            }

            if (block === Pacman.PILL) {
                game.eatenPill();
            }
        }

        return {
            "new" : position,
            "old" : oldPosition
        };
    };

    function isMidSquare(x) {
        var rem = x % 10;
        return rem > 3 || rem < 7;
    };

    function calcAngle(dir, pos) {
        if (dir == RIGHT && (pos.x % 10 < 5)) {
            return {"start":0.25, "end":1.75, "direction": false};
        } else if (dir === DOWN && (pos.y % 10 < 5)) {
            return {"start":0.75, "end":2.25, "direction": false};
        } else if (dir === UP && (pos.y % 10 < 5)) {
            return {"start":1.25, "end":1.75, "direction": true};
        } else if (dir === LEFT && (pos.x % 10 < 5)) {
            return {"start":0.75, "end":1.25, "direction": true};
        }
```

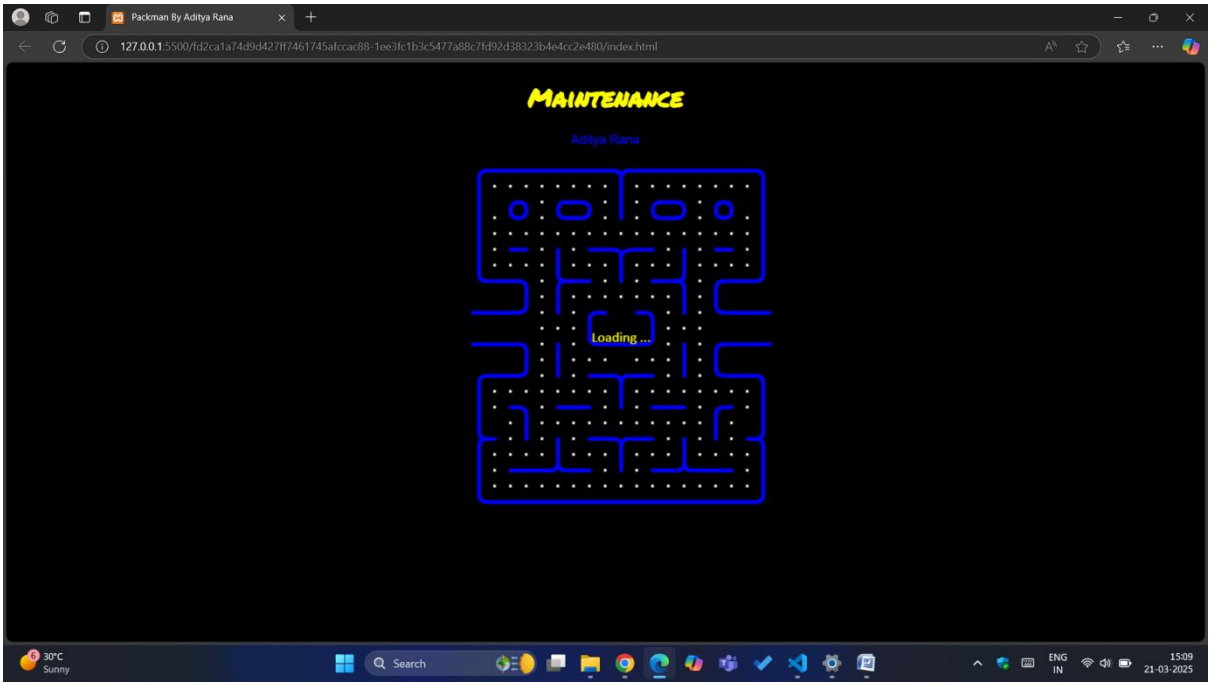
```
        return {"start":0, "end":2, "direction": false};
    };

    function drawDead(ctx, amount) {

        var size = map.blockSize,
            half = size / 2;

        if (amount >= 1) {
            return;
        }

        ctx.fillStyle = "#FFFF00";
        ctx.beginPath();
        ctx.moveTo(((position.x/10) * size) + half,
                   ((position.y/10) * size) + half);

        ctx.arc(((position.x/10) * size) + half,
                ((position.y/10) * size) + half,
                half, 0, Math.PI * 2 * amount, true);

        ctx.fill();
    };

    function draw(ctx) {

        var s     = map.blockSize,
            angle = calcAngle(direction, position);

        ctx.fillStyle = "#FFFF00";

        ctx.beginPath();

        ctx.moveTo(((position.x/10) * s) + s / 2,
                   ((position.y/10) * s) + s / 2);

        ctx.arc(((position.x/10) * s) + s / 2,
                ((position.y/10) * s) + s / 2,
                s / 2, Math.PI * angle.start,
                Math.PI * angle.end, angle.direction);

        ctx.fill();
    };

    initUser();

    return {
        "draw"          : draw,
        "drawDead"      : drawDead,
        "loseLife"      : loseLife,
        "getLives"      : getLives,
        "score"         : score,
        "addScore"      : addScore,
        "theScore"      : theScore,
        "keyDown"       : keyDown,
        "move"          : move,
        "newLevel"      : newLevel,
        "reset"         : reset,
        "resetPosition" : resetPosition
    };
};
```

```
Pacman.Map = function (size) {

    var height    = null,
        width     = null,
        blockSize = size,
        pillSize  = 0,
        map       = null;

    function withinBounds(y, x) {
        return y >= 0 && y < height && x >= 0 && x < width;
    }

    function isWall(pos) {
        return withinBounds(pos.y, pos.x) && map[pos.y][pos.x] === Pacman.WALL;
    }

    function isFloorSpace(pos) {
        if (!withinBounds(pos.y, pos.x)) {
            return false;
        }
        var peice = map[pos.y][pos.x];
        return peice === Pacman.EMPTY ||
            peice === Pacman.BISCUIT ||
            peice === Pacman.PILL;
    }

    function drawWall(ctx) {

        var i, j, p, line;

        ctx.strokeStyle = "#0000FF";
        ctx.lineWidth   = 5;
        ctx.lineCap     = "round";

        for (i = 0; i < Pacman.WALLS.length; i += 1) {
            line = Pacman.WALLS[i];
            ctx.beginPath();

            for (j = 0; j < line.length; j += 1) {

                p = line[j];

                if (p.move) {
                    ctx.moveTo(p.move[0] * blockSize, p.move[1] * blockSize);
                } else if (p.line) {
                    ctx.lineTo(p.line[0] * blockSize, p.line[1] * blockSize);
                } else if (p.curve) {
                    ctx.quadraticCurveTo(p.curve[0] * blockSize,
                                         p.curve[1] * blockSize,
                                         p.curve[2] * blockSize,
                                         p.curve[3] * blockSize);
                }
            }
            ctx.stroke();
        }
    }

    function reset() {
        map    = Pacman.MAP.clone();
        height = map.length;
```

```
        width  = map[0].length;
    };

            startNewGame();
        } else if (e.keyCode === KEY.S) {
            audio.disableSound();
            localStorage["soundDisabled"] = !soundDisabled();
        } else if (e.keyCode === KEY.P && state === PAUSE) {
            audio.resume();
            map.draw(ctx);
            setState(stored);
        } else if (e.keyCode === KEY.P) {
            stored = state;
            setState(PAUSE);
            audio.pause();
            map.draw(ctx);
            dialog("Paused");
        } else if (state !== PAUSE) {
            return user.keyDown(e);
        }
        return true;
    }

    function loseLife() {
        setState(WAITING);
        user.loseLife();
        if (user.getLives() > 0) {
            startLevel();
        }
    }

    function setState(nState) {
        state = nState;
        stateChanged = true;
    };

    function collided(user, ghost) {
        return (Math.sqrt(Math.pow(ghost.x - user.x, 2) +
                          Math.pow(ghost.y - user.y, 2))) < 10;
    };

    function drawFooter() {

        var topLeft  = (map.height * map.blockSize),
            textBase = topLeft + 17;

        ctx.fillStyle = "#000000";
        ctx.fillRect(0, topLeft, (map.width * map.blockSize), 30);

        ctx.fillStyle = "#FFFF00";

        for (var i = 0, len = user.getLives(); i < len; i++) {
            ctx.fillStyle = "#FFFF00";
            ctx.beginPath();
            ctx.moveTo(150 + (25 * i) + map.blockSize / 2,
                       (topLeft+1) + map.blockSize / 2);

            ctx.arc(150 + (25 * i) + map.blockSize / 2,
                    (topLeft+1) + map.blockSize / 2,
                    map.blockSize / 2, Math.PI * 0.25, Math.PI * 1.75, false);
            ctx.fill();
```

```
        }

        ctx.fillStyle = !soundDisabled() ? "#00FF00" : "#FF0000";
        ctx.font = "bold 16px sans-serif";



Object.prototype.clone = function () {
    var i, newObj = (this instanceof Array) ? [] : {};
    for (i in this) {
        if (i === 'clone') {
            continue;
        }
        if (this[i] && typeof this[i] === "object") {
            newObj[i] = this[i].clone();
        } else {
            newObj[i] = this[i];
        }
    }
    return newObj;
};

$(function(){
  var el = document.getElementById("pacman");

  if (Modernizr.canvas && Modernizr.localstorage &&
      Modernizr.audio && (Modernizr.audio.ogg || Modernizr.audio.mp3)) {
    window.setTimeout(function () { PACMAN.init(el,
"https://raw.githubusercontent.com/daleharvey/pacman/master/"); }, 0);
  } else {
    el.innerHTML = "Sorry, needs a decent browser<br /><small>" +
      "(firefox 3.6+, Chrome 4+, Opera 10+ and Safari 4+)</small>";
  }
});



    </script>
</body>
</html>
```

Output:


Packman By Aditya Rana
127.0.0.1:5500/fd2ca1a74d9d427ff7461745afccac88-1ee3fc1b3c5477a88c7fd92d38323b4e4cc2e480/index.html
MAINTENANCE
Aditya Rana
Loading ...
30°C
Sunny
Search
ENG
IN
15:09
21-03-2025