```
<!DOCTYP
E html>
            <html>
            <head>
                   <title>Maintenance - Moota.co</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;">Sedang migrasi server dan upgrade
            engine 23:00 s/d 05:00. Mohon doanya :)</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"></scr
ipt>
       <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)
        };
    };


    /* Collision detection(walls) is done when a ghost lands on an
     * exact block, make sure they dont skip over it
     */
    function addBounded(x1, x2) {
```

```
    var rem    = x1 % 10,
        result = rem + x2;
    if (rem !== 0 && result > 10) {
        return x1 + (10 - rem);
    } else if(rem > 0 && result < 0) {
        return x1 - rem;
    }
    return x1 + x2;
};

function isVunerable() {
    return eatable !== null;
};

function isDangerous() {
    return eaten === null;
};


function isHidden() {
    return eatable === null && eaten !== null;
};

function getRandomDirection() {
    var moves = (direction === LEFT || direction === RIGHT)
        ? [UP, DOWN] : [LEFT, RIGHT];
    return moves[Math.floor(Math.random() * 2)];
};

function reset() {
    eaten = null;
    eatable = null;
    position = {"x": 90, "y": 80};
    direction = getRandomDirection();
    due = getRandomDirection();
};

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

function oppositeDirection(dir) {
    return dir === LEFT && RIGHT ||
        dir === RIGHT && LEFT ||
        dir === UP && DOWN || UP;
};


function makeEatable() {
    direction = oppositeDirection(direction);
    eatable = game.getTick();
};


function eat() {
    eatable = null;
    eaten = game.getTick();
};
```

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


function secondsAgo(tick) {
    return (game.getTick() - tick) / Pacman.FPS;
};


function getColour() {
    if (eatable) {
        if (secondsAgo(eatable) > 5) {
            return game.getTick() % 20 > 10 ? "#FFFFFF" : "#0000BB";
        } else {
            return "#0000BB";
        }
    } else if(eaten) {
        return "#222";
    }
    return colour;
};


function draw(ctx) {

    var s    = map.blockSize,
        top  = (position.y/10) * s,
        left = (position.x/10) * s;

    if (eatable && secondsAgo(eatable) > 8) {
        eatable = null;
    }

    if (eaten && secondsAgo(eaten) > 3) {
        eaten = null;
    }

    var tl = left + s;
    var base = top + s - 3;
    var inc = s / 10;
```

```
    var high = game.getTick() % 10 > 5 ? 3  : -3;
    var low  = game.getTick() % 10 > 5 ? -3 : 3;


    ctx.fillStyle = getColour();
    ctx.beginPath();


    ctx.moveTo(left, base);


    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) {

        npos = getNewCoord(due, position);

        if (onGrid &&
            map.isFloorSpace({
                "y":pointToCoord(nextSquare(npos.y, due)),
                "x":pointToCoord(nextSquare(npos.x, due))})) {
            direction = due;
        } else {
            npos = null;
        }
    }

    if (npos === null) {
        npos = getNewCoord(direction, position);
    }

    if (onGrid &&
        map.isWallSpace({
            "y" : pointToCoord(nextSquare(npos.y, direction)),
            "x" : pointToCoord(nextSquare(npos.x, direction))
        })) {

        due = getRandomDirection();
        return move(ctx);
    }


    position = npos;

    var tmp = pane(position);
    if (tmp) {
        position = tmp;
    }

    due = getRandomDirection();

    return {
        "new" : position,
        "old" : oldPos
    };
};
```

```
    return {
        "eat"         : eat,
        "isVunerable" : isVunerable,
        "isDangerous" : isDangerous,
        "makeEatable" : makeEatable,
        "reset"       : reset,
        "move"        : move,
        "draw"        : draw
    };
};


Pacman.User = function (game, map) {

    var position  = null,
        direction = null,
        eaten     = null,
        due       = null,
        lives     = null,
        score     = 5,
        keyMap    = {};

    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;
    };
```

```
function block(pos) {
    return map[pos.y][pos.x];
};

function setBlock(pos, type) {
    map[pos.y][pos.x] = type;
};


function drawPills(ctx) {


    if (++pillSize > 30) {
        pillSize = 0;
    }

    for (i = 0; i < height; i += 1) {
              for (j = 0; j < width; j += 1) {
            if (map[i][j] === Pacman.PILL) {
                ctx.beginPath();


                ctx.fillStyle = "#000";
                       ctx.fillRect((j * blockSize), (i * blockSize),
                              blockSize, blockSize);


                ctx.fillStyle = "#FFF";
                ctx.arc((j * blockSize) + blockSize / 2,
                        (i * blockSize) + blockSize / 2,
                        Math.abs(5 - (pillSize/3)),
                        0,
                        Math.PI * 2, false);
                ctx.fill();
                ctx.closePath();
            }
              }
       }
};

function draw(ctx) {

    var i, j, size = blockSize;


    ctx.fillStyle = "#000";
       ctx.fillRect(0, 0, width * size, height * size);


    drawWall(ctx);

    for (i = 0; i < height; i += 1) {
              for (j = 0; j < width; j += 1) {
                     drawBlock(i, j, ctx);
              }
       }
};

function drawBlock(y, x, ctx) {
```

```
        var layout = map[y][x];


        if (layout === Pacman.PILL) {
            return;
        }


        ctx.beginPath();

        if (layout === Pacman.EMPTY || layout === Pacman.BLOCK ||
            layout === Pacman.BISCUIT) {

            ctx.fillStyle = "#000";
                   ctx.fillRect((x * blockSize), (y * blockSize),
                          blockSize, blockSize);


            if (layout === Pacman.BISCUIT) {
                ctx.fillStyle = "#FFF";
                       ctx.fillRect((x * blockSize) + (blockSize / 2.5),
                              (y * blockSize) + (blockSize / 2.5),
                              blockSize / 6, blockSize / 6);
                   }
        }
        ctx.closePath();
    };


    reset();

    return {
        "draw"         : draw,
        "drawBlock"    : drawBlock,
        "drawPills"    : drawPills,
        "block"        : block,
        "setBlock"     : setBlock,
        "reset"        : reset,
        "isWallSpace"  : isWall,
        "isFloorSpace" : isFloorSpace,
        "height"       : height,
        "width"        : width,
        "blockSize"    : blockSize
    };
};


Pacman.Audio = function(game) {

    var files          = [],
        endEvents      = [],
        progressEvents = [],
        playing        = [];

    function load(name, path, cb) {


        var f = files[name] = document.createElement("audio");
```

```
        progressEvents[name] = function(event) { progress(event, name, cb);
};

        f.addEventListener("canplaythrough", progressEvents[name], true);
        f.setAttribute("preload", "true");
        f.setAttribute("autobuffer", "true");
        f.setAttribute("src", path);
        f.pause();
    };


    function progress(event, name, callback) {
        if (event.loaded === event.total && typeof callback === "function") {
            callback();
            files[name].removeEventListener("canplaythrough",
                                            progressEvents[name], true);
        }
    };


    function disableSound() {
        for (var i = 0; i < playing.length; i++) {
            files[playing[i]].pause();
            files[playing[i]].currentTime = 0;
        }
        playing = [];
    };


    function ended(name) {


        var i, tmp = [], found = false;


        files[name].removeEventListener("ended", endEvents[name], true);


        for (i = 0; i < playing.length; i++) {
            if (!found && playing[i]) {
                found = true;
            } else {
                tmp.push(playing[i]);
            }
        }
        playing = tmp;
    };


    function play(name) {
        if (!game.soundDisabled()) {
            endEvents[name] = function() { ended(name); };
            playing.push(name);
            files[name].addEventListener("ended", endEvents[name], true);
            files[name].play();
        }
    };
```

```
    function pause() {
        for (var i = 0; i < playing.length; i++) {
            files[playing[i]].pause();
        }
    };

    function resume() {
        for (var i = 0; i < playing.length; i++) {
            files[playing[i]].play();
        }
    };

    return {
        "disableSound" : disableSound,
        "load"         : load,
        "play"         : play,
        "pause"        : pause,
        "resume"       : resume
    };
};


var PACMAN = (function () {


    var state        = WAITING,
        audio        = null,
        ghosts       = [],
        ghostSpecs   = ["#00FFDE", "#FF0000", "#FFB8DE", "#FFB847"],
        eatenCount   = 0,
        level        = 0,
        tick         = 0,
        ghostPos, userPos,
        stateChanged = true,
        timerStart   = null,
        lastTime     = 0,
        ctx          = null,
        timer        = null,
        map          = null,
        user         = null,
        stored       = null;


    function getTick() {
        return tick;
    };


    function drawScore(text, position) {
        ctx.fillStyle = "#FFFFFF";
        ctx.font      = "12px BDCartoonShoutRegular";
        ctx.fillText(text,
                     (position["new"]["x"] / 10) * map.blockSize,
                     ((position["new"]["y"] + 5) / 10) * map.blockSize);
    }

    function dialog(text) {
        ctx.fillStyle = "#FFFF00";
```

```
    ctx.font      = "18px Calibri";
    var width = ctx.measureText(text).width,
        x     = ((map.width * map.blockSize) - width) / 2;
    ctx.fillText(text, x, (map.height * 10) + 8);
}


function soundDisabled() {
    return localStorage["soundDisabled"] === "true";
};

function startLevel() {
    user.resetPosition();
    for (var i = 0; i < ghosts.length; i += 1) {
        ghosts[i].reset();
    }
    audio.play("start");
    timerStart = tick;
    setState(COUNTDOWN);
}


function startNewGame() {
    setState(WAITING);
    level = 1;
    user.reset();
    map.reset();
    map.draw(ctx);
    startLevel();
}


function keyDown(e) {
    if (e.keyCode === KEY.N) {
        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";
        //ctx.fillText("♪", 10, textBase);
        ctx.fillText("s", 10, textBase);


        ctx.fillStyle = "#FFFF00";
        ctx.font      = "14px Calibri";
        ctx.fillText("Score: " + user.theScore(), 30, textBase);
        ctx.fillText("Level: " + level, 260, textBase);
    }


    function redrawBlock(pos) {
        map.drawBlock(Math.floor(pos.y/10), Math.floor(pos.x/10), ctx);
        map.drawBlock(Math.ceil(pos.y/10), Math.ceil(pos.x/10), ctx);
    }


    function mainDraw() {
```

```
    var diff, u, i, len, nScore;

    ghostPos = [];


    for (i = 0, len = ghosts.length; i < len; i += 1) {
        ghostPos.push(ghosts[i].move(ctx));
    }
    u = user.move(ctx);

    for (i = 0, len = ghosts.length; i < len; i += 1) {
        redrawBlock(ghostPos[i].old);
    }
    redrawBlock(u.old);

    for (i = 0, len = ghosts.length; i < len; i += 1) {
        ghosts[i].draw(ctx);
    }
    user.draw(ctx);

    userPos = u["new"];

    for (i = 0, len = ghosts.length; i < len; i += 1) {
        if (collided(userPos, ghostPos[i]["new"])) {
            if (ghosts[i].isVunerable()) {
                audio.play("eatghost");
                ghosts[i].eat();
                eatenCount += 1;
                nScore = eatenCount * 50;
                drawScore(nScore, ghostPos[i]);
                user.addScore(nScore);
                setState(EATEN_PAUSE);
                timerStart = tick;
            } else if (ghosts[i].isDangerous()) {
                audio.play("die");
                setState(DYING);
                timerStart = tick;
            }
        }
    }
};


function mainLoop() {


    var diff;


    if (state !== PAUSE) {
        ++tick;
    }


    map.drawPills(ctx);


    if (state === PLAYING) {
```

```
            mainDraw();
        } else if (state === WAITING && stateChanged) {
            stateChanged = false;
            map.draw(ctx);
            dialog("Press N to start a New game");
        } else if (state === EATEN_PAUSE &&
                   (tick - timerStart) > (Pacman.FPS / 3)) {
            map.draw(ctx);
            setState(PLAYING);
        } else if (state === DYING) {
            if (tick - timerStart > (Pacman.FPS * 2)) {
                loseLife();
            } else {
                redrawBlock(userPos);
                for (i = 0, len = ghosts.length; i < len; i += 1) {
                    redrawBlock(ghostPos[i].old);
                    ghostPos.push(ghosts[i].draw(ctx));
                }
                user.drawDead(ctx, (tick - timerStart) / (Pacman.FPS * 2));
            }
        } else if (state === COUNTDOWN) {

            diff = 5 + Math.floor((timerStart - tick) / Pacman.FPS);

            if (diff === 0) {
                map.draw(ctx);
                setState(PLAYING);
            } else {
                if (diff !== lastTime) {
                    lastTime = diff;
                    map.draw(ctx);
                    dialog("Starting in: " + diff);
                }
            }
        }


        drawFooter();
    }


    function eatenPill() {
        audio.play("eatpill");
        timerStart = tick;
        eatenCount = 0;
        for (i = 0; i < ghosts.length; i += 1) {
            ghosts[i].makeEatable(ctx);
        }
    };

    function completedLevel() {
        setState(WAITING);
        level += 1;
        map.reset();
        user.newLevel();
        startLevel();
    };


    function keyPress(e) {
```

```
        if (state !== WAITING && state !== PAUSE) {
            e.preventDefault();
            e.stopPropagation();
        }
    };

    function init(wrapper, root) {

        var i, len, ghost,
            blockSize = wrapper.offsetWidth / 19,
            canvas    = document.createElement("canvas");

        canvas.setAttribute("width", (blockSize * 19) + "px");
        canvas.setAttribute("height", (blockSize * 22) + 30 + "px");


        wrapper.appendChild(canvas);


        ctx  = canvas.getContext('2d');


        audio = new Pacman.Audio({"soundDisabled":soundDisabled});
        map   = new Pacman.Map(blockSize);
        user  = new Pacman.User({
            "completedLevel" : completedLevel,
            "eatenPill"      : eatenPill
        }, map);


        for (i = 0, len = ghostSpecs.length; i < len; i += 1) {
            ghost = new Pacman.Ghost({"getTick":getTick}, map,
ghostSpecs[i]);
            ghosts.push(ghost);
        }

        map.draw(ctx);
        dialog("Loading ...");


        var extension = Modernizr.audio.ogg ? 'ogg' : 'mp3';


        var audio_files = [
            ["start", root + "audio/opening_song." + extension],
            ["die", root + "audio/die." + extension],
            ["eatghost", root + "audio/eatghost." + extension],
            ["eatpill", root + "audio/eatpill." + extension],
            ["eating", root + "audio/eating.short." + extension],
            ["eating2", root + "audio/eating.short." + extension]
        ];


        load(audio_files, function() { loaded(); });
    };


    function load(arr, callback) {
```

```
        if (arr.length === 0) {
            callback();
        } else {
            var x = arr.pop();
            audio.load(x[0], x[1], function() { load(arr, callback); });
        }
    };

    function loaded() {


        dialog("Press N to Start");

        document.addEventListener("keydown", keyDown, true);
        document.addEventListener("keypress", keyPress, true);

        timer = window.setInterval(mainLoop, 1000 / Pacman.FPS);
    };

    return {
        "init" : init
    };

}());


/* Human readable keyCode index */
var KEY = {'BACKSPACE': 8, 'TAB': 9, 'NUM_PAD_CLEAR': 12, 'ENTER': 13,
'SHIFT': 16, 'CTRL': 17, 'ALT': 18, 'PAUSE': 19, 'CAPS_LOCK': 20, 'ESCAPE':
27, 'SPACEBAR': 32, 'PAGE_UP': 33, 'PAGE_DOWN': 34, 'END': 35, 'HOME': 36,
'ARROW_LEFT': 37, 'ARROW_UP': 38, 'ARROW_RIGHT': 39, 'ARROW_DOWN': 40,
'PRINT_SCREEN': 44, 'INSERT': 45, 'DELETE': 46, 'SEMICOLON': 59,
'WINDOWS_LEFT': 91, 'WINDOWS_RIGHT': 92, 'SELECT': 93, 'NUM_PAD_ASTERISK':
106, 'NUM_PAD_PLUS_SIGN': 107, 'NUM_PAD_HYPHEN-MINUS': 109,
'NUM_PAD_FULL_STOP': 110, 'NUM_PAD_SOLIDUS': 111, 'NUM_LOCK': 144,
'SCROLL_LOCK': 145, 'SEMICOLON': 186, 'EQUALS_SIGN': 187, 'COMMA': 188,
'HYPHEN-MINUS': 189, 'FULL_STOP': 190, 'SOLIDUS': 191, 'GRAVE_ACCENT': 192,
'LEFT_SQUARE_BRACKET': 219, 'REVERSE_SOLIDUS': 220, 'RIGHT_SQUARE_BRACKET':
221, 'APOSTROPHE': 222};


(function () {
       /* 0 - 9 */
       for (var i = 48; i <= 57; i++) {
        KEY['' + (i - 48)] = i;
       }
       /* A - Z */
       for (i = 65; i <= 90; i++) {
        KEY['' + String.fromCharCode(i)] = i;
       }
       /* NUM_PAD_0 - NUM_PAD_9 */
       for (i = 96; i <= 105; i++) {
        KEY['NUM_PAD_' + (i - 96)] = i;
       }
       /* F1 - F12 */
       for (i = 112; i <= 123; i++) {
        KEY['F' + (i - 112 + 1)] = i;
       }
})();
```

```
Pacman.WALL    = 0;
Pacman.BISCUIT = 1;
Pacman.EMPTY   = 2;
Pacman.BLOCK   = 3;
Pacman.PILL    = 4;


Pacman.MAP = [
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0],
       [0, 4, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 4, 0],
       [0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0],
       [0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0],
       [0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0],
       [0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0],
       [0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0],
       [2, 2, 2, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 2, 2, 2],
       [0, 0, 0, 0, 1, 0, 1, 0, 0, 3, 0, 0, 1, 0, 1, 0, 0, 0, 0],
       [2, 2, 2, 2, 1, 1, 1, 0, 3, 3, 3, 0, 1, 1, 1, 2, 2, 2, 2],
       [0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0],
       [2, 2, 2, 0, 1, 0, 1, 1, 1, 2, 1, 1, 1, 0, 1, 0, 2, 2, 2],
       [0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0],
       [0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0],
       [0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0],
       [0, 4, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 4, 0],
       [0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0],
       [0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0],
       [0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0],
       [0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
];


Pacman.WALLS = [

    [{"move": [0, 9.5]}, {"line": [3, 9.5]},
     {"curve": [3.5, 9.5, 3.5, 9]}, {"line": [3.5, 8]},
     {"curve": [3.5, 7.5, 3, 7.5]}, {"line": [1, 7.5]},
     {"curve": [0.5, 7.5, 0.5, 7]}, {"line": [0.5, 1]},
     {"curve": [0.5, 0.5, 1, 0.5]}, {"line": [9, 0.5]},
     {"curve": [9.5, 0.5, 9.5, 1]}, {"line": [9.5, 3.5]}],


    [{"move": [9.5, 1]},
     {"curve": [9.5, 0.5, 10, 0.5]}, {"line": [18, 0.5]},
     {"curve": [18.5, 0.5, 18.5, 1]}, {"line": [18.5, 7]},
     {"curve": [18.5, 7.5, 18, 7.5]}, {"line": [16, 7.5]},
     {"curve": [15.5, 7.5, 15.5, 8]}, {"line": [15.5, 9]},
     {"curve": [15.5, 9.5, 16, 9.5]}, {"line": [19, 9.5]}],


    [{"move": [2.5, 5.5]}, {"line": [3.5, 5.5]}],


    [{"move": [3, 2.5]},
     {"curve": [3.5, 2.5, 3.5, 3]},
     {"curve": [3.5, 3.5, 3, 3.5]},
     {"curve": [2.5, 3.5, 2.5, 3]},
     {"curve": [2.5, 2.5, 3, 2.5]}],
```

```
    [{"move": [15.5, 5.5]}, {"line": [16.5, 5.5]}],


    [{"move": [16, 2.5]}, {"curve": [16.5, 2.5, 16.5, 3]},
     {"curve": [16.5, 3.5, 16, 3.5]}, {"curve": [15.5, 3.5, 15.5, 3]},
     {"curve": [15.5, 2.5, 16, 2.5]}],


    [{"move": [6, 2.5]}, {"line": [7, 2.5]}, {"curve": [7.5, 2.5, 7.5, 3]},
     {"curve": [7.5, 3.5, 7, 3.5]}, {"line": [6, 3.5]},
     {"curve": [5.5, 3.5, 5.5, 3]}, {"curve": [5.5, 2.5, 6, 2.5]}],


    [{"move": [12, 2.5]}, {"line": [13, 2.5]}, {"curve": [13.5, 2.5, 13.5,
3]},
     {"curve": [13.5, 3.5, 13, 3.5]}, {"line": [12, 3.5]},
     {"curve": [11.5, 3.5, 11.5, 3]}, {"curve": [11.5, 2.5, 12, 2.5]}],


    [{"move": [7.5, 5.5]}, {"line": [9, 5.5]}, {"curve": [9.5, 5.5, 9.5, 6]},
     {"line": [9.5, 7.5]}],
    [{"move": [9.5, 6]}, {"curve": [9.5, 5.5, 10.5, 5.5]},
     {"line": [11.5, 5.5]}],




    [{"move": [5.5, 5.5]}, {"line": [5.5, 7]}, {"curve": [5.5, 7.5, 6, 7.5]},
     {"line": [7.5, 7.5]}],
    [{"move": [6, 7.5]}, {"curve": [5.5, 7.5, 5.5, 8]}, {"line": [5.5,
9.5]}],


    [{"move": [13.5, 5.5]}, {"line": [13.5, 7]},
     {"curve": [13.5, 7.5, 13, 7.5]}, {"line": [11.5, 7.5]}],
    [{"move": [13, 7.5]}, {"curve": [13.5, 7.5, 13.5, 8]},
     {"line": [13.5, 9.5]}],


    [{"move": [0, 11.5]}, {"line": [3, 11.5]}, {"curve": [3.5, 11.5, 3.5,
12]},
     {"line": [3.5, 13]}, {"curve": [3.5, 13.5, 3, 13.5]}, {"line": [1,
13.5]},
     {"curve": [0.5, 13.5, 0.5, 14]}, {"line": [0.5, 17]},
     {"curve": [0.5, 17.5, 1, 17.5]}, {"line": [1.5, 17.5]}],
    [{"move": [1, 17.5]}, {"curve": [0.5, 17.5, 0.5, 18]}, {"line": [0.5,
21]},
     {"curve": [0.5, 21.5, 1, 21.5]}, {"line": [18, 21.5]},
     {"curve": [18.5, 21.5, 18.5, 21]}, {"line": [18.5, 18]},
     {"curve": [18.5, 17.5, 18, 17.5]}, {"line": [17.5, 17.5]}],
    [{"move": [18, 17.5]}, {"curve": [18.5, 17.5, 18.5, 17]},
     {"line": [18.5, 14]}, {"curve": [18.5, 13.5, 18, 13.5]},
     {"line": [16, 13.5]}, {"curve": [15.5, 13.5, 15.5, 13]},
     {"line": [15.5, 12]}, {"curve": [15.5, 11.5, 16, 11.5]},
     {"line": [19, 11.5]}],
```

```
    [{"move": [5.5, 11.5]}, {"line": [5.5, 13.5]}],
    [{"move": [13.5, 11.5]}, {"line": [13.5, 13.5]}],


    [{"move": [2.5, 15.5]}, {"line": [3, 15.5]},
     {"curve": [3.5, 15.5, 3.5, 16]}, {"line": [3.5, 17.5]}],
    [{"move": [16.5, 15.5]}, {"line": [16, 15.5]},
     {"curve": [15.5, 15.5, 15.5, 16]}, {"line": [15.5, 17.5]}],


    [{"move": [5.5, 15.5]}, {"line": [7.5, 15.5]}],
    [{"move": [11.5, 15.5]}, {"line": [13.5, 15.5]}],

    [{"move": [2.5, 19.5]}, {"line": [5, 19.5]},
     {"curve": [5.5, 19.5, 5.5, 19]}, {"line": [5.5, 17.5]}],
    [{"move": [5.5, 19]}, {"curve": [5.5, 19.5, 6, 19.5]},
     {"line": [7.5, 19.5]}],


    [{"move": [11.5, 19.5]}, {"line": [13, 19.5]},
     {"curve": [13.5, 19.5, 13.5, 19]}, {"line": [13.5, 17.5]}],
    [{"move": [13.5, 19]}, {"curve": [13.5, 19.5, 14, 19.5]},
     {"line": [16.5, 19.5]}],


    [{"move": [7.5, 13.5]}, {"line": [9, 13.5]},
     {"curve": [9.5, 13.5, 9.5, 14]}, {"line": [9.5, 15.5]}],
    [{"move": [9.5, 14]}, {"curve": [9.5, 13.5, 10, 13.5]},
     {"line": [11.5, 13.5]}],


    [{"move": [7.5, 17.5]}, {"line": [9, 17.5]},
     {"curve": [9.5, 17.5, 9.5, 18]}, {"line": [9.5, 19.5]}],
    [{"move": [9.5, 18]}, {"curve": [9.5, 17.5, 10, 17.5]},
     {"line": [11.5, 17.5]}],


    [{"move": [8.5, 9.5]}, {"line": [8, 9.5]}, {"curve": [7.5, 9.5, 7.5,
10]},
     {"line": [7.5, 11]}, {"curve": [7.5, 11.5, 8, 11.5]},
     {"line": [11, 11.5]}, {"curve": [11.5, 11.5, 11.5, 11]},
     {"line": [11.5, 10]}, {"curve": [11.5, 9.5, 11, 9.5]},
     {"line": [10.5, 9.5]}]
];


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>
```