Project Report

Operating System

Ludo Game

Group Members:

- Abdullah Bin Faiz (19i-0695 – CS-4A)
- Saroosh Hammad (19i-0599 – CS-4A)

Submission Date: 15th June, 2021

Pseudo Code for phase 1:

Stored the whole interface of Ludo in string.

Broke the string line by line and stored it in a 2d char array.

Manually calculated the x and y coordinates of players' home where all tokens are placed initially.

From 2d char array, by knowing the coordinates of players' home, placed all the tokens on the interface of game.

```
struct Coordinate {
    int x;
    int y;
    Coordinate() {}
    Coordinate(int x_, int y_): x(x_), y(y_) {}
};

//Positions function sets tokens of each player
void positions() {
    // a check to see if a token is at home or not
    for (from 0 to 3) {
        for (from 0 to no of tokens{
            if (token of player is not free) {
                if (token of player is not clear) {
                    players[i].tokens[j].coords.x = homeCoords[k].x;
                    players[i].tokens[j].coords.y = homeCoords[k].y;
                } else {
                    players[i].tokens[j].coords.x = clearCoords[i][5].x;
                    players[i].tokens[j].coords.y = clearCoords[i][5].y;
                }
```

```
                    }
                }
                k += 4-numberOfTokens;
            }

            // setting all of the tokens for position
            for (from 0 to 3) {
                for (from 0 to no of tokens {
                    if (player is dissqualified== false)
                        boardCoords[y coordinates of the tokens of player][ x coordinates of the tokens of player] = player token at display;
                    else {
                        for (from 0 to no of tokens) {
                            coordinates of tokens of player = players starting point;
                        }
                        boardCoords[y coordinate of player][x coordinate of player] = ' ';
                    }
                }
            }
        }
        void drawBoard() {
            // compile the board into the string
            board = "";
         board += ".................................................................................";
            board += ".                              .    .    .    .                              .";
            board += ".                              ......+-----/*****\\                              .";
            board += ".                              .    |    |    | <--                            .";
            board += ".                    /*****\\-----\\*****/                    .";
            board += ".                    |    |    |    .                    .";
            board += ".      . . . .       \\*****/-----|......      . . . .      .";
            board += ".                    .    |    |    .                    .";
            board += ".                    ......|-----|......                    .";
            board += ".                    .    |    |    .                    .";
            board += ".    |               ......|-----|......                    .";
            board += ".    V                .    |    |    .                    .";
            board += "....../*****\\......................+-----------------+................./*****\\............";
            board += ".   |    |    .    .    .    |            |    .    .    |    |    .    .";
            board += "......\\*****/-----------------------+            +-----------------\\*****/-----
        +......";
            board += ".   |    |    |    |    |    |    CLEAR    |    |    |    |    |    |    .";
            board += "......+-----/*****\\-----------------+            +-----------------------
        /*****\\......";
            board += ".   .    |    |    .    .    |            |    .    .    .    |    |    .";
            board += "............\\*****/.................+-----------------+.......................\\*****/......";
            board += ".                    .    |    |    .                 ^      .";
            board += ".                    ......|-----|......               |      .";
            board += ".                    .    |    |    .                    .";
            board += ".                    ......|-----|......                    .";
            board += ".                    .    |    |    .                    .";
            board += ".      . . . .       ......|-----/*****\\      . . . .      .";
            board += ".                    .    |    |    |                    .";
            board += ".                    /*****\\-----\\*****/                    .";
            board += ".                 --> |    |    |    .                    .";
            board += ".                    \\*****/-----+......                    .";
            board += ".                    .    .    .    .                    .";
            board += "..........................................................................................";
            // transfer it over to the coordinates list
            for (from 0 t o30) {
                    for (from 0 to 90) {
                            boardCoords[i][j] = board[k];
                    }
            }
            cout << "CheckBlocks\n";
            // call function for checking block states of each position player is on
            checkBlocks();

            cout << "CheckPositions\n";
            // this is where the tokens will have their positions set before the displaying of the
        board
            positions();

            cout << "CheckPrint\n";
            for (0 to 30) {
                    for (0 to 90{
                            cout << boardCoords[i][j];
                    }
                    cout << endl;
            }
        }
```

Pseudo Code for phase 2:

```
struct Path { // the outer loop of the board (circular linked list)
        Space* head;
```

```
Path(): head(NULL) {}


void insert(Coordinate values) {
        Space* newSpace = new Space;
        newSpace coords = values;
        newSpace position = pos values;
        if (pos_values is equal 2 OR pos_values is equal 10 OR pos_values is equal 15 OR
pos_values is equal 23 OR pos_values is equal 28 OR pos_values is equal 36 OR pos_values is equal 41
OR pos_values is equal 49) {
        newSpace isSafe = true;
        } else { newSpace isSafe = false; } // just for safe measures
        newSpace blockFormed = false;
        newSpace->blockedBy = 0;
        pos_values++;
        if (head equals NULL) {
                head = newSpace;
                head->next = head;
        } else if (head->next equals head) {
                head->next = newSpace;
                newSpace->next = head;
        } else {
                Space* traversal = head;
                while (traversal next not equal to head) {
                        traversal = traversal to next;
                }
                traversal->next = newSpace;
                newSpace->next = head;
        }
}


bool isSpaceSafe(int position) { // for use in crash function
```

```
        Space* traversal = head;
        while (traversal->position not equal to position) { traversal = traversal->next; }
        return traversal->isSafe;
}

void setBlockState(bool blockState, int player, int position) {
        Space* traversal = head;
        while (traversal->position not equals position) {
                traversal = traversal->next;
        }
        traversal->setBlockState(blockState, player);
}

bool getBlockState(int position) { // for use in locak moveToken function
        Space* traversal = head;
        do {
                if (position equals traversal->position) break;
                traversal = traversal->next;
        } while (traversal not equals head);
        return traversal->getBlockState();
}

int getBlockedBy(int position) { // for use in locak moveToken function
        Space* traversal = head;
        do {
                if (position equals traversal->position) break;
                traversal = traversal->next;
        } while (traversal not equals head);
        return traversal->getBlockedBy();
}
```

```
        Coordinate moveToken(int& roll, int position, int hitRecord, int homePosition, int player, bool&
block, bool& isInside, int slot) {
                Space* traversal = head;
                int i = 0;
                while (i less than position) {
                        traversal = traversal->next;
                        i++;
                }
                Coordinate currCoords;
                currCoords.x = traversal->coords.x;
                currCoords.y = traversal->coords.y;
                while (roll no equals 0) {
                        if (traversal->next->next->position equals homePosition) {
                                if (hitRecord reater than 0) {
                                        currCoords.x = clearCoords[playerIndex-1][0].x;
                                        currCoords.y = clearCoords[playerIndex-1][0].y;
                                        isInside = true;
                                        return currCoords;
                                }
                        }
                        traversal = traversal->next;
                        if (traversal->getBlockState() equals true) {
                                if (traversal->getBlockedBy() not equals player) {
                                        cout << "Movement is not possible; returning token to
position...";
                                        block = true;
                                        sleep(3);
                                        return currCoords;
                                }
                        }
                        Decrement roll
                }
                cout << "New position: " << traversal->position << endl;
                Coordinate sendCoords;
                sendCoords.x = traversal->coords.x;
                sendCoords.y = traversal->coords.y;
                return sendCoords;
        }

        void display() {
                Space* traversal = head;
                do {
                        display x and y coordinates
                        traversal = traversal->next;
                } while (traversal not equals head);
        }
};
void init() { // a function to initialize everything needed
        // first initialize the home corodinates
        for (0 to 16) {
                if (i less than 4) {
                        homeCoords[i].x = 13 + (i%4 * 3);
                        homeCoords[i].y = 6;
                } else if (i less than 8) {
                        homeCoords[i].x = 67 + (i%4 * 3);
                        homeCoords[i].y = 6;
                } else if (i lesss than 12) {
                        homeCoords[i].x = 13 + (i%4 * 3);
                        homeCoords[i].y = 24;
                } else if (i less than 16) {
                        homeCoords[i].x = 67 + (i%4 * 3);
                        homeCoords[i].y = 24;
                }
        }

        // initialize pathSpaces with pre-assigned coordinate values for the board
        for (0 to 52) {
                if (i equals 0) { pathSpaces[i].y = 15; pathSpaces[i].x = 1; }
                else if (i equals 1) { pathSpaces[i].y = 13; pathSpaces[i].x = 1; }
                else if (i equals 2) { pathSpaces[i].y = 13; pathSpaces[i].x = 7; }
                else if (I equals 3) { pathSpaces[i].y = 13; pathSpaces[i].x = 13; }
                else if (i equals 4) { pathSpaces[i].y = 13; pathSpaces[i].x = 19; }
                else if (i equals 5) { pathSpaces[i].y = 13; pathSpaces[i].x = 25; }
                else if (i equals 6) { pathSpaces[i].y = 13; pathSpaces[i].x = 31; }
                else if (i equals 7) { pathSpaces[i].y = 11; pathSpaces[i].x = 37; }
                else if (i equals 8) { pathSpaces[i].y = 9; pathSpaces[i].x = 37; }
                else if (i equals 9) { pathSpaces[i].y = 7; pathSpaces[i].x = 37; }
                else if (i equals 10) { pathSpaces[i].y = 5; pathSpaces[i].x = 37; }
                else if (i equals 11) { pathSpaces[i].y = 3; pathSpaces[i].x = 37; }
                else if (i equals 12) { pathSpaces[i].y = 1; pathSpaces[i].x = 37; }
                else if (i equals 13) { pathSpaces[i].y = 1; pathSpaces[i].x = 43; }
                else if (i equals 14) { pathSpaces[i].y = 1; pathSpaces[i].x = 49; }
                else if (i equals 15) { pathSpaces[i].y = 3; pathSpaces[i].x = 49; }
                else if (i equals 16) { pathSpaces[i].y = 5; pathSpaces[i].x = 49; }
                else if (i equals 17) { pathSpaces[i].y = 7; pathSpaces[i].x = 49; }
                else if (i equals 18) { pathSpaces[i].y = 9; pathSpaces[i].x = 49; }
                else if (i equals 19) { pathSpaces[i].y = 11; pathSpaces[i].x = 49; }
                else if (i equals 20) { pathSpaces[i].y = 13; pathSpaces[i].x = 55; }
                else if (i equals 21) { pathSpaces[i].y = 13; pathSpaces[i].x = 61; }
                else if (i equals 22) { pathSpaces[i].y = 13; pathSpaces[i].x = 67; }
```

```
            else if (i equals 23) { pathSpaces[i].y = 13; pathSpaces[i].x = 73; }
            else if (i equals 24) { pathSpaces[i].y = 13; pathSpaces[i].x = 79; }
            else if (i equals 25) { pathSpaces[i].y = 13; pathSpaces[i].x = 85; }
            else if (i equals 26) { pathSpaces[i].y = 15; pathSpaces[i].x = 85; }
            else if (i equals 27) { pathSpaces[i].y = 17; pathSpaces[i].x = 85; }
            else if (i equals 28) { pathSpaces[i].y = 17; pathSpaces[i].x = 79; }
            else if (i equals 29) { pathSpaces[i].y = 17; pathSpaces[i].x = 73; }
            else if (i equals 30) { pathSpaces[i].y = 17; pathSpaces[i].x = 67; }
            else if (i equals 31) { pathSpaces[i].y = 17; pathSpaces[i].x = 61; }
            else if (i equals 32) { pathSpaces[i].y = 17; pathSpaces[i].x = 55; }
            else if (i equals 33) { pathSpaces[i].y = 19; pathSpaces[i].x = 49; }
            else if (i equals 34) { pathSpaces[i].y = 21; pathSpaces[i].x = 49; }
            else if (i equals 35) { pathSpaces[i].y = 23; pathSpaces[i].x = 49; }
            else if (i equals 36) { pathSpaces[i].y = 25; pathSpaces[i].x = 49; }
            else if (i equals 37) { pathSpaces[i].y = 27; pathSpaces[i].x = 49; }
            else if (i equals 38) { pathSpaces[i].y = 29; pathSpaces[i].x = 49; }
            else if (i equals 39) { pathSpaces[i].y = 29; pathSpaces[i].x = 43; }
            else if (i equals 40) { pathSpaces[i].y = 29; pathSpaces[i].x = 37; }
            else if (i equals 41) { pathSpaces[i].y = 27; pathSpaces[i].x = 37; }
            else if (i equals 42) { pathSpaces[i].y = 25; pathSpaces[i].x = 37; }
            else if (i equals 43) { pathSpaces[i].y = 23; pathSpaces[i].x = 37; }
            else if (i equals 44) { pathSpaces[i].y = 21; pathSpaces[i].x = 37; }
            else if (i equals 45) { pathSpaces[i].y = 19; pathSpaces[i].x = 37; }
            else if (i equals 46) { pathSpaces[i].y = 17; pathSpaces[i].x = 31; }
            else if (i equals 47) { pathSpaces[i].y = 17; pathSpaces[i].x = 25; }
            else if (i equals 48) { pathSpaces[i].y = 17; pathSpaces[i].x = 19; }
            else if (i equals 49) { pathSpaces[i].y = 17; pathSpaces[i].x = 13; }
            else if (i equals 50) { pathSpaces[i].y = 17; pathSpaces[i].x = 7; }
            else if (i equals 51) { pathSpaces[i].y = 17; pathSpaces[i].x = 1; }
        }
```

```
// initialize the clear spaces' coordinates for each player
for (0 to 3) {
    if (i equals 0) {
        clearCoords[i][0].x = 7; clearCoords[i][0].y = 15; clearCoords[i][0].position = 52;
        clearCoords[i][1].x = 13; clearCoords[i][1].y = 15; clearCoords[i][1].position = 53;
        clearCoords[i][2].x = 19; clearCoords[i][2].y = 15; clearCoords[i][2].position = 54;
        clearCoords[i][3].x = 25; clearCoords[i][3].y = 15; clearCoords[i][3].position = 55;
        clearCoords[i][4].x = 31; clearCoords[i][4].y = 15; clearCoords[i][4].position = 56;
        clearCoords[i][5].x = 37; clearCoords[i][5].y = 15; clearCoords[i][5].position = 57;
    }
    else if (i equals 1) {
        clearCoords[i][0].x = 43; clearCoords[i][0].y = 3; clearCoords[i][0].position = 58;
        clearCoords[i][1].x = 43; clearCoords[i][1].y = 5; clearCoords[i][1].position = 59;
        clearCoords[i][2].x = 43; clearCoords[i][2].y = 7; clearCoords[i][2].position = 60;
        clearCoords[i][3].x = 43; clearCoords[i][3].y = 9; clearCoords[i][3].position = 61;
        clearCoords[i][4].x = 43; clearCoords[i][4].y = 11; clearCoords[i][4].position = 62;
        clearCoords[i][5].x = 43; clearCoords[i][5].y = 13; clearCoords[i][5].position = 63;
    }
    else if (i equals 2) {
        clearCoords[i][0].x = 79; clearCoords[i][0].y = 15; clearCoords[i][0].position = 64;
```

```
                clearCoords[i][1].x = 73; clearCoords[i][1].y = 15; clearCoords[i][1].position = 65;
                clearCoords[i][2].x = 67; clearCoords[i][2].y = 15; clearCoords[i][2].position = 66;
                clearCoords[i][3].x = 61; clearCoords[i][3].y = 15; clearCoords[i][3].position = 67;
                clearCoords[i][4].x = 55; clearCoords[i][4].y = 15; clearCoords[i][4].position = 68;
                clearCoords[i][5].x = 49; clearCoords[i][5].y = 15; clearCoords[i][5].position = 69;
            }
            else if (i equals 3) {
                clearCoords[i][0].x = 43; clearCoords[i][0].y = 27; clearCoords[i][0].position = 70;
                clearCoords[i][1].x = 43; clearCoords[i][1].y = 25; clearCoords[i][1].position = 71;
                clearCoords[i][2].x = 43; clearCoords[i][2].y = 23; clearCoords[i][2].position = 72;
                clearCoords[i][3].x = 43; clearCoords[i][3].y = 21; clearCoords[i][3].position = 73;
                clearCoords[i][4].x = 43; clearCoords[i][4].y = 19; clearCoords[i][4].position = 74;
                clearCoords[i][5].x = 43; clearCoords[i][5].y = 17; clearCoords[i][5].position = 75;
            }
        }
void crash( currTokenPosition) {
    for (0 to 3) {
        if (i not equals  playerIndex-1) {
            for (0 to no of tokens) {
                if (currTokenPosition equals players[i].tokens[j].position) {
                    if (players[i].tokens[j].isFree equals true) {
                        bool isItSafeSpace = false;
                        for (0 to 7) {
```

```
                                        if (players[i].tokens[j].position equals
safePositions[k]) {
                                                  isItSafeSpace = true;
                                                  break;
                                             }
                                        }
                                        if (not safe space) {
                                             players[i].tokens[j].isFree = false;
                                             players[playerIndex-1].hitRecord increment;
                                             players[playerIndex-1].noSixOccurences = 0;
                                             hitInRound = true;
                                             someoneHitAToken = true;
                                             cout << "Hit!\n";
                                             checkAllTokens();
                                        } else {
                                             cout << "Space is safe from crashes.\n";
                                        }
                                   }
                              }
                         }
                    }
          }
}

void checkAllTokens() {
          for (0 to 3 {
                    bool allLocked = true;
                    for (0 to no of tokens {
                              if (players[i].tokens[j].isFree equals true) {
                                        allLocked = false;
```

```
                break;
            }
        }
        if (allLocked) {
            players[i].hasFreeToken = false;
        }
    }
}

void movement() {

    char input;
    int turn = 1;
    bool repeat = true;
    /* movement START */
    while (repeat) {

        repeat = false;
        cout << "Player " << playerIndex << ", turn " << turn << ": Number of times no sixes
rolled: " << players[playerIndex-1].noSixOccurences << endl;
        cout << "Enter D to roll the dice: ";
        do { cin >> input; } while (input not equals 'd' && input not equals 'D');
        int roll, tokenToMove;
        if (input equals 'd' || input equals 'D') {
            roll = getRoll();
            //if (equals1) { if (roll equals 6) roll--; else roll = 6; }
            cout << "Roll obtained: " << roll << endl;
            if (roll == 6) {
                repeat = true;
                players[playerIndex-1].noSixOccurences = 0;
```

```
if (players[playerIndex-1].hasFreeToken equals false) {
    cout << "One token has been released!\n";
    initialTokens[turn-1] = players[playerIndex-1].tokens[0];
    players[playerIndex-1].tokens[0].isFree = true;
    players[playerIndex-1].tokens[0].coords.x = players[playerIndex-1].startingPoint.x + players[playerIndex-1].tokens[0].slot;
    players[playerIndex-1].tokens[0].coords.y = players[playerIndex-1].startingPoint.y;
    players[playerIndex-1].tokens[0].position = players[playerIndex-1].startingPosition;
    players[playerIndex-1].hasFreeToken = true;
} else {
    bool allFree = true;
    for (0 to no of tokens {
        if (players[playerIndex-1].tokens[j].isFree equals false && players[playerIndex-1].tokens[j].isClear equals false) {
            allFree = false;
            break;
        }
    }
    if (allFree equals false) {
        int option;
        cout << "You have a choice!\nEnter 1 to release new token; enter 2 to move from current tokens: ";
        do { cin >> option; } while (option != 1 && option != 2);
        if (option equals 1) {
            int j = 0;
            while (players[playerIndex-1].tokens[j].isFree == true) { j++; cout << "Skipped to " << j+1 << endl; }
            initialTokens[turn-1] = players[playerIndex-1].tokens[j];
            players[playerIndex-1].tokens[j].isFree = true;
```

```
                                        players[playerIndex-1].tokens[j].coords.x =
players[playerIndex-1].startingPoint.x + players[playerIndex-1].tokens[j].slot;
                                        players[playerIndex-1].tokens[j].coords.y =
players[playerIndex-1].startingPoint.y;
                                        players[playerIndex-1].tokens[j].position =
players[playerIndex-1].startingPosition;
                                    } else if (option == 2) {
                                        cout << "Choose a token to move:\n";
                                        for (0 to no of tokens) {
                                            if (players[playerIndex-
1].tokens[j].isFree == true && players[playerIndex-1].tokens[j].isClear tokens false) {
                                                cout << "Token " << j+1 <<
endl;
                                            }
                                        }
                                        cout << "Enter: ";
                                        do { cin >> tokenToMove; } while
(players[playerIndex-1].tokens[tokenToMove-1].isFree == false);
                                        initialTokens[turn-1] = players[playerIndex-
1].tokens[tokenToMove-1];
                                        bool insideState = false;
                                        int val = roll;
                                        if (players[playerIndex-1].tokens[tokenToMove-
1].isInside equals false) {
                                            bool blockState = false;
                                            Coordinate getCoords =
path.moveToken(val, players[playerIndex-1].tokens[tokenToMove-1].position, players[playerIndex-
1].hitRecord, players[playerIndex-1].startingPosition, playerIndex, blockState, insideState,
tokenToMove-1);
                                            players[playerIndex-
1].tokens[tokenToMove-1].coords.x = getCoords.x + players[playerIndex-1].tokens[tokenToMove-
1].slot;
                                            players[playerIndex-
1].tokens[tokenToMove-1].coords.y = getCoords.y;
```

```
                                        if (val equals 0) {
                                                if (!blockState) {
                                                        int steps = roll;
                                                        while (steps != 0) {

        players[playerIndex-1].tokens[tokenToMove-1].position++;
                                                                if
(players[playerIndex-1].tokens[tokenToMove-1].position == 52)

        players[playerIndex-1].tokens[tokenToMove-1].position = 0;
                                                                steps--;
                                                        }
                                                } else {
                                                        players[playerIndex-
1].tokens[tokenToMove-1] = initialTokens[turn-1];
                                                }
                                                crash(players[playerIndex-
1].tokens[tokenToMove-1].position);
                                        }
                                }
                                if (insideState equals true || players[playerIndex-
1].tokens[tokenToMove-1].isInside) {
                                        cout << "RETURNED\n";
                                        if (players[playerIndex-
1].tokens[tokenToMove-1].isInside equals false) {
                                                players[playerIndex-
1].tokens[tokenToMove-1].position = clearCoords[playerIndex-1][0].position;
                                                cout << players[playerIndex-
1].tokens[tokenToMove-1].position << endl;
                                        }
                                        cout << "Second check\n";
                                        players[playerIndex-
1].tokens[tokenToMove-1].isInside = true;
                                        cout << "One more\n";
                                        pathToClear(val, players[playerIndex-
1].tokens[tokenToMove-1].position, tokenToMove-1);
                                        cout << "Success!\n";
                                    }
                                    //players[i-1].tokens[tokenToMove-1].position =
getPos(players[i-1].tokens[tokenToMove-1].coords);
                                    //sleep(2);
                                }
                            } else {
                                cout << "Choose a token to move:\n";
                                for (no of tokens {
                                    if (players[playerIndex-1].tokens[j].isFree
equals true && players[playerIndex-1].tokens[j].isClear equals false) {
                                        cout << "Token " << j+1 << endl;
                                    }
                                }
                                cout << "Enter: ";
                                do { cin >> tokenToMove; } while (players[playerIndex-
1].tokens[tokenToMove-1].isFree == false);
                                initialTokens[turn-1] = players[playerIndex-
1].tokens[tokenToMove-1];
                                bool insideState = false;
                                int val = roll;
                                if (players[playerIndex-1].tokens[tokenToMove-
1].isInside equals false) {
                                    bool blockState = false;
                                    Coordinate getCoords = path.moveToken(val,
players[playerIndex-1].tokens[tokenToMove-1].position, players[playerIndex-1].hitRecord,
players[playerIndex-1].startingPosition, playerIndex, blockState, insideState, tokenToMove-1);
                                    players[playerIndex-1].tokens[tokenToMove-
1].coords.x = getCoords.x + players[playerIndex-1].tokens[tokenToMove-1].slot;
                                    players[playerIndex-1].tokens[tokenToMove-
1].coords.y = getCoords.y;
```

```
                                        if (val equals 0) {
                                            if (not blockState) {
                                                int steps = roll;
                                                while (steps not equals 0) {
                                                    players[playerIndex-
1].tokens[tokenToMove-1].position++;
                                                    if (players[playerIndex-
1].tokens[tokenToMove-1].position == 52)

    players[playerIndex-1].tokens[tokenToMove-1].position = 0;
                                                    steps decrement;
                                                }
                                            } else {
                                                players[playerIndex-
1].tokens[tokenToMove-1] = initialTokens[turn-1];
                                            }
                                            crash(players[playerIndex-
1].tokens[tokenToMove-1].position);
                                        }
                                    }
                                    if (insideState == true || players[playerIndex-
1].tokens[tokenToMove-1].isInside)  {
                                        cout << "RETURNED\n";
                                        if (players[playerIndex-1].tokens[tokenToMove-
1].isInside equals false) {
                                            players[playerIndex-
1].tokens[tokenToMove-1].position = clearCoords[playerIndex-1][0].position;
                                            cout << "Home pos: " <<
players[playerIndex-1].tokens[tokenToMove-1].position << endl;
                                        }
                                        cout << "Second check\n";
                                        players[playerIndex-1].tokens[tokenToMove-
1].isInside = true;
                                        cout << "One more\n";
```

```
                                                pathToClear(val, players[playerIndex-
1].tokens[tokenToMove-1].position, tokenToMove-1);
                                                cout << "Success!\n";
                                        }
                                        //players[i-1].tokens[tokenToMove-1].position =
getPos(players[i-1].tokens[tokenToMove-1].coords);
                                        //sleep(2);
                                }
                        }
                } else {
                        if (players[playerIndex-1].hasFreeToken equals false) {
                                cout << "Cannot play; passing turn to next player.\n";
                        } else {
                                cout << "Choose a token to move:\n";
                                for (int j = 0; j < numberOfTokens; j++) {
                                        if (players[playerIndex-1].tokens[j].isFree == true &&
players[playerIndex-1].tokens[j].isClear == false) {
                                                cout << "Token " << j+1 << endl;
                                        }
                                }
                                cout << "Enter: ";
                                do {
                                        tokenToMove = -1;
                                        if (tokenToMove < 1 || tokenToMove > 4) { cin >>
tokenToMove; }
                                } while (players[playerIndex-1].tokens[tokenToMove-1].isFree
== false);
                                initialTokens[turn-1] = players[playerIndex-
1].tokens[tokenToMove-1];
                                bool insideState = false;
                                int val = roll;
```

```
if (players[playerIndex-1].tokens[tokenToMove-1].isInside == false) {
    bool blockState = false;
    Coordinate getCoords = path.moveToken(val, players[playerIndex-1].tokens[tokenToMove-1].position, players[playerIndex-1].hitRecord, players[playerIndex-1].startingPosition, playerIndex, blockState, insideState, tokenToMove-1);
    players[playerIndex-1].tokens[tokenToMove-1].coords.x = getCoords.x + players[playerIndex-1].tokens[tokenToMove-1].slot;
    players[playerIndex-1].tokens[tokenToMove-1].coords.y = getCoords.y;


    if (val equals 0) {
        if (!blockState) {
            int steps = roll;
            while (steps != 0) {
                players[playerIndex-1].tokens[tokenToMove-1].position++;
                if (players[playerIndex-1].tokens[tokenToMove-1].position == 52)
                    players[playerIndex-1].tokens[tokenToMove-1].position = 0;
                steps--;
            }
        } else {
            players[playerIndex-1].tokens[tokenToMove-1] = initialTokens[turn-1];
        }
        crash(players[playerIndex-1].tokens[tokenToMove-1].position);
    }
}
if (insideState equals true || players[playerIndex-1].tokens[tokenToMove-1].isInside) {
    cout << "RETURNED\n";
```

```
                                                if (players[playerIndex-1].tokens[tokenToMove-
1].isInside == false) {
                                                        players[playerIndex-1].tokens[tokenToMove-
1].position = clearCoords[playerIndex-1][0].position;
                                                        cout << players[playerIndex-
1].tokens[tokenToMove-1].position << endl;
                                                }
                                                cout << "Second check\n";
                                                players[playerIndex-1].tokens[tokenToMove-1].isInside
= true;
                                                cout << "One more\n";
                                                pathToClear(val, players[playerIndex-
1].tokens[tokenToMove-1].position, tokenToMove-1);
                                                cout << "Success!\n";
                                        }
                                        //players[i-1].tokens[tokenToMove-1].position =
getPos(players[i-1].tokens[tokenToMove-1].coords);
                                        //sleep(2);
                                }
                        }


                }
                turn++;
                if (roll != 6 && someoneHitAToken == false) {
                        players[playerIndex-1].noSixOccurences++;
                        turn = 1;
                        repeat = false;
                }
                if (someoneHitAToken equals true) someoneHitAToken = false;
                if (turn == 4) {
                        cout << "3 sixes rolled! Skipping to next player... "; //Returning every token
moved to their positions...";
```

```
                    // for (int j = 0; j < 3; j++) {
                    //          players[i-1].tokens[initialTokens[j].slot] = initialTokens[j];
                    // }
                    turn = 1;
                    repeat = false;
            }

            for (int j = 0; j < numberOfTokens; j++) {
                    if (players[playerIndex-1].tokens[j].position == clearCoords[playerIndex-
1][5].position) {
                            cout << "Token " << players[playerIndex-1].tokens[j].slot+1 << " of
player " << playerIndex << " has reached home!\n";
                            players[playerIndex-1].tokens[j].isClear = true;
                            players[playerIndex-1].tokens[j].isFree = false;
                    }
            }

    }

void* playerThread(void* arg) {
        int oldState, oldType;
        pthread_setcancelstate(PTHREAD_CANCEL_ENABLE, &oldState);
        pthread_setcanceltype(PTHREAD_CANCEL_ASYNCHRONOUS, &oldType);
        while (true) {

                bool allClear = true;
                for (no of tokens{
                        if (players[playerIndex-1].tokens[j].isClear equals false) {
                                allClear = false;
                                break;
                        }
                }
                if (allClear equals true) {
                        players[playerIndex-1].hasWon = true;
                        whoWon = playerIndex;
                        someoneWonArr[playerIndex-1] = true;
                        someoneWon = true;
                }

                while (hasCompletedTask[playerIndex-1] || players[playerIndex-1].disqualified equals
true || someoneWonArr[playerIndex-1] == true) {
                        playerIndex increment
                        if (playerIndex  greater than 4) playerIndex = 1;
                }

                sem_wait(&semaphore);
                movement();
                sem_post(&semaphore);

                sem_wait(&semaphore);
                drawBoard();
                hasCompletedTask[playerIndex-1] = 1;
                sem_post(&semaphore);
        }
        pthread_exit(NULL);
}

void* masterThreadFunc(void* arg) {

        void* status[4];
```

```
for (0 to 3) {
        pthread_create(&playerThreads[i], NULL, playerThread, NULL);
}

while (true) {

        if (closeGame equals true)
                break;

        if (players[playerIndex-1].disqualified equals true) {
                players[playerIndex-1].winState = 2; // means it lost
                hasCompletedTask[playerIndex-1] = 2;
                pthread_cancel(playerThreads[playerIndex]);
                cout << "Cancelled thread\n";
        }

        bool someoneStillLeft = false;
        for (0 to 3 {
                if (someoneWonArr[i] equals false) { someoneStillLeft = true; break; }
        }

        if (someoneStillLeft equals false) {
                cout << "All players have finished the game.\n";
                break;
        }

        if (someoneWon) {
                cout << "\nPlayer---" << whoWon << "---has won the game!\n";
                pthread_cancel(playerThreads[whoWon-1]);
                someoneWon = false;
```

```
            }
            if (hasCompletedTask[0] and hasCompletedTask[1] and hasCompletedTask[2] and
hasCompletedTask[3]) {
                  if (hasCompletedTask[0] equals 1) hasCompletedTask[0] = 0;
                  if (hasCompletedTask[1] equals 1) hasCompletedTask[1] = 0;
                  if (hasCompletedTask[2] equals 1) hasCompletedTask[2] = 0;
                  if (hasCompletedTask[3] equals 1) hasCompletedTask[3] = 0;
                  cout << "---\nRound End\n---\n";
                  if (hitInRound) {
                        cout << "Hit records:\n";
                        for (0 to 3 {
                              cout << "Player " << j+1 << " hit count: " <<
players[j].hitRecord << endl;
                        }
                        hitInRound = false;
                  }
            }
      }
      pthread_exit((void*)true);
}
```

Operating Systems concepts implemented in this project:

We implemented the following concepts of OS:

- Threads (to allow for controlling of multiple players at the same time)
- Synchronization (in the form of semaphores; to maintain order in which players play)

Implemented Code:

```
#include <iostream>
#include <unistd.h>
#include <semaphore.h>
#include <pthread.h>
```

```
#include <string.h>

#include <time.h>

#include <stdlib.h>

#include <sys/wait.h>

using namespace std;


int numberOfTokens, pos_values = 0; // for initializing number of tokens and spaces' positions
respectively

int playerIndex = 1; // global index to keep track of which player is being taken care of; starting from first
player

int hasCompletedTask[4] = {0, 0, 0, 0}; // for threads to check if their task has been done so that they can
repeat

// the condition is that 0 means it's still doing its task, 1 means it has done its task, and 2 means it is
knocked out

sem_t semaphore; // semaphore lock designed to support serialization

pthread_t masterThread, playerThreads[4]; // a master thread and 4 player threads calling functions

bool hitInRound = false; // check for if any player has hit a token; to show results after each round

int playerWhoHit; // check wich player was it who crashed an opposing team's token

bool closeGame = false, someoneHitAToken = false; // booleans designed to close the game and check if
someone crashed another token


bool someoneWonArr[4] = {false, false, false, false}; // array to check who has won

bool someoneWon = false; // failsafe boolean to show who has won

int whoWon = -1; // failsale index showing which player has won


// coordinate of each space of outer Ludo loop

struct Coordinate {

        int x;

        int y;

        Coordinate() {}

        Coordinate(int x_, int y_): x(x_), y(y_) {}

};
```

```
// special coordinates designed for the last clear steps going to CLEAR home
struct CoordinateClear {
	int x;
	int y;
	int position;
	CoordinateClear() {}
	CoordinateClear(int x_, int y_): x(x_), y(y_), position(0) {}
};

CoordinateClear clearCoords[4][6];
// index 0 will have player 1's coordinates, index 1 player 2's, index 2 player 4's, and index 3 player 3's

struct Space { // a space on the board
	Coordinate coords;
	int position; // check current position on the board; unique for every space
	bool isSafe; // default not safe; check if the space token is on is safe for it to not be eliminated
	bool blockFormed; // in case two tokens of the same colour end up on the same position, a block
forms which no other opponents can land on or pass
	int blockedBy; // another variation to check which player has blocked the space
	Space* next; // linked list pointing to next space of outer Ludo loop

	void setBlockState(bool blockState, int player) {
		blockFormed = blockState;
		blockedBy = player;
	}

	bool getBlockState() { // for use in locak moveToken function
		bool retVal = blockFormed;
		return retVal;
```

```
        }

        int getBlockedBy() { // for use in locak moveToken function
                int retVal = blockedBy;
                return retVal;
        }
};

struct Path { // the outer loop of the board (circular linked list)
        Space* head;

        Path(): head(NULL) {}

        void insert(Coordinate values) {
                Space* newSpace = new Space;
                newSpace->coords = values;
                newSpace->position = pos_values;
                if (pos_values == 2 || pos_values == 10 || pos_values == 15 || pos_values == 23 ||
pos_values == 28 || pos_values == 36 || pos_values == 41 || pos_values == 49) {
                        newSpace->isSafe = true;
                } else { newSpace->isSafe = false; } // just for safe measures
                newSpace->blockFormed = false;
                newSpace->blockedBy = 0;
                pos_values++;
                if (head == NULL) {
                        head = newSpace;
                        head->next = head;
                } else if (head->next == head) {
                        head->next = newSpace;
                        newSpace->next = head;
        } else {
            Space* traversal = head;
            while (traversal->next != head) {
                traversal = traversal->next;
            }
            traversal->next = newSpace;
            newSpace->next = head;
        }
    }

    bool isSpaceSafe(int position) { // for use in crash function
        Space* traversal = head;
        while (traversal->position != position) { traversal = traversal->next; }
        return traversal->isSafe;
    }

    void setBlockState(bool blockState, int player, int position) {
        Space* traversal = head;
        while (traversal->position != position) {
            traversal = traversal->next;
        }
        traversal->setBlockState(blockState, player);
    }

    bool getBlockState(int position) { // for use in local moveToken function
        Space* traversal = head;
        do {
            if (position == traversal->position) break;
            traversal = traversal->next;
        } while (traversal != head);
        return traversal->getBlockState();
    }

    int getBlockedBy(int position) { // for use in local moveToken function
        Space* traversal = head;
        do {
            if (position == traversal->position) break;
            traversal = traversal->next;
        } while (traversal != head);
        return traversal->getBlockedBy();
    }

    Coordinate moveToken(int& roll, int position, int hitRecord, int homePosition, int player, bool&
block, bool& isInside, int slot) {
        Space* traversal = head;
        int i = 0;
        while (i < position) {
            traversal = traversal->next;
            i++;
        }
        Coordinate currCoords;
        currCoords.x = traversal->coords.x;
        currCoords.y = traversal->coords.y;
        while (roll != 0) {
            if (traversal->next->next->position == homePosition) {
                if (hitRecord > 0) {
                    currCoords.x = clearCoords[playerIndex-1][0].x;
                    currCoords.y = clearCoords[playerIndex-1][0].y;
                    isInside = true;
                    return currCoords;
                }
            }
            traversal = traversal->next;
            if (traversal->getBlockState() == true) {
                if (traversal->getBlockedBy() != player) {
                    cout << "Movement is not possible; returning token to
position...";
                    block = true;
                    return currCoords;
                }
            }
            roll = roll - 1;
        }
        Coordinate sendCoords;
        sendCoords.x = traversal->coords.x;
        sendCoords.y = traversal->coords.y;
        return sendCoords;
    }

    void display() { // temporary function to see if spaces loaded correctly
        Space* traversal = head;
        do {
            cout << traversal->coords.x << ", " << traversal->coords.y << endl;
            traversal = traversal->next;
        } while (traversal != head);
    }
};

Path path;
```

```
struct Token {
            char display; // add an ASCII to symbolize the tokens
            Coordinate coords; // store a token's current coordinates in here
   bool isFree = false; // checks if player rolls a 6 then isFree turns true for one of the token which is then
released onto the board
            bool isClear = false; // condition to check if the token has reached CLEAR space
            int slot; // position on the board so that each token has their respective slot to occupy
            int position; // what space the token is currently on
            bool isInside = false; // check for keeping track of token inside clear spaces
};

Token initialTokens[3];

struct Player {
   string name; // ask the player for their name
            bool hasFreeToken = false;
            Coordinate startingPoint; // the coordinates of where the player's tokens will start
            int startingPosition; // space where tokens of that player first land upon release
   int numberOfTokens; // store number of tokens as value (ranging from 1 to 4)
   //Player* isTeamedWith;
            int hitRecord = 0; // check for how many times a player has crashed another opponent's token
   Token* tokens; // we'll initialize this on the basis of players' input;
            int noSixOccurences = 0; // if this reaches 20, the player is kicked out
            bool disqualified = false; // check dependent on above variable
            bool hasWon = false; // condition to check if all tokens of player have gone to CLEAR space
            int winState = 0; // state to check current status of player; 0 means still playing, 1 means won, 2
means lost
};

Player* players; // pointer to players, initialized dynamically
```

```
string board; // our entire board's layout will be drawn in the form of a string first

char boardCoords[31][91] = {}; // all of the contents of the board in the above string will be stored inside
this array for display

// now this array will have divisions for the tokens to be placed on

// instead of making a char array, we'll use a structure containing the x and y values of the string/board


Coordinate pathSpaces[52]; // coordinates of outer loop of Ludo to include in circular linked list


int safePositions[8] = {2, 10, 15, 23, 28, 36, 41, 49}; // failsafe to check if token is on safe spot


Coordinate homeCoords[16]; // 4 for each player

// player 1 will be in ranges 13-22/6, player 2 will be in ranges 67-76/6

// player 3 will be in ranges 13-22/24, player 4 will be in ranges 76-76/24


void init() { // a function to initialize everything needed

        // first initialize the home corodinates

        for (int i = 0; i < 16; i++) {

                if (i < 4) {

                        homeCoords[i].x = 13 + (i%4 * 3);

                        homeCoords[i].y = 6;

                } else if (i < 8) {

                        homeCoords[i].x = 67 + (i%4 * 3);

                        homeCoords[i].y = 6;

                } else if (i < 12) {

                        homeCoords[i].x = 13 + (i%4 * 3);

                        homeCoords[i].y = 24;

                } else if (i < 16) {

                        homeCoords[i].x = 67 + (i%4 * 3);

                        homeCoords[i].y = 24;

                }
```

```
}

// initialize pathSpaces with pre-assigned coordinate values for the board
for (int i = 0; i < 52; i++) {
        if (i == 0) { pathSpaces[i].y = 15; pathSpaces[i].x = 1; }
        else if (i == 1) { pathSpaces[i].y = 13; pathSpaces[i].x = 1; }
        else if (i == 2) { pathSpaces[i].y = 13; pathSpaces[i].x = 7; }
        else if (i == 3) { pathSpaces[i].y = 13; pathSpaces[i].x = 13; }
        else if (i == 4) { pathSpaces[i].y = 13; pathSpaces[i].x = 19; }
        else if (i == 5) { pathSpaces[i].y = 13; pathSpaces[i].x = 25; }
        else if (i == 6) { pathSpaces[i].y = 13; pathSpaces[i].x = 31; }
        else if (i == 7) { pathSpaces[i].y = 11; pathSpaces[i].x = 37; }
        else if (i == 8) { pathSpaces[i].y = 9; pathSpaces[i].x = 37; }
        else if (i == 9) { pathSpaces[i].y = 7; pathSpaces[i].x = 37; }
        else if (i == 10) { pathSpaces[i].y = 5; pathSpaces[i].x = 37; }
        else if (i == 11) { pathSpaces[i].y = 3; pathSpaces[i].x = 37; }
        else if (i == 12) { pathSpaces[i].y = 1; pathSpaces[i].x = 37; }
        else if (i == 13) { pathSpaces[i].y = 1; pathSpaces[i].x = 43; }
        else if (i == 14) { pathSpaces[i].y = 1; pathSpaces[i].x = 49; }
        else if (i == 15) { pathSpaces[i].y = 3; pathSpaces[i].x = 49; }
        else if (i == 16) { pathSpaces[i].y = 5; pathSpaces[i].x = 49; }
        else if (i == 17) { pathSpaces[i].y = 7; pathSpaces[i].x = 49; }
        else if (i == 18) { pathSpaces[i].y = 9; pathSpaces[i].x = 49; }
        else if (i == 19) { pathSpaces[i].y = 11; pathSpaces[i].x = 49; }
        else if (i == 20) { pathSpaces[i].y = 13; pathSpaces[i].x = 55; }
        else if (i == 21) { pathSpaces[i].y = 13; pathSpaces[i].x = 61; }
        else if (i == 22) { pathSpaces[i].y = 13; pathSpaces[i].x = 67; }
        else if (i == 23) { pathSpaces[i].y = 13; pathSpaces[i].x = 73; }
        else if (i == 24) { pathSpaces[i].y = 13; pathSpaces[i].x = 79; }
        else if (i == 25) { pathSpaces[i].y = 13; pathSpaces[i].x = 85; }
        else if (i == 26) { pathSpaces[i].y = 15; pathSpaces[i].x = 85; }
        else if (i == 27) { pathSpaces[i].y = 17; pathSpaces[i].x = 85; }
        else if (i == 28) { pathSpaces[i].y = 17; pathSpaces[i].x = 79; }
        else if (i == 29) { pathSpaces[i].y = 17; pathSpaces[i].x = 73; }
        else if (i == 30) { pathSpaces[i].y = 17; pathSpaces[i].x = 67; }
        else if (i == 31) { pathSpaces[i].y = 17; pathSpaces[i].x = 61; }
        else if (i == 32) { pathSpaces[i].y = 17; pathSpaces[i].x = 55; }
        else if (i == 33) { pathSpaces[i].y = 19; pathSpaces[i].x = 49; }
        else if (i == 34) { pathSpaces[i].y = 21; pathSpaces[i].x = 49; }
        else if (i == 35) { pathSpaces[i].y = 23; pathSpaces[i].x = 49; }
        else if (i == 36) { pathSpaces[i].y = 25; pathSpaces[i].x = 49; }
        else if (i == 37) { pathSpaces[i].y = 27; pathSpaces[i].x = 49; }
        else if (i == 38) { pathSpaces[i].y = 29; pathSpaces[i].x = 49; }
        else if (i == 39) { pathSpaces[i].y = 29; pathSpaces[i].x = 43; }
        else if (i == 40) { pathSpaces[i].y = 29; pathSpaces[i].x = 37; }
        else if (i == 41) { pathSpaces[i].y = 27; pathSpaces[i].x = 37; }
        else if (i == 42) { pathSpaces[i].y = 25; pathSpaces[i].x = 37; }
        else if (i == 43) { pathSpaces[i].y = 23; pathSpaces[i].x = 37; }
        else if (i == 44) { pathSpaces[i].y = 21; pathSpaces[i].x = 37; }
        else if (i == 45) { pathSpaces[i].y = 19; pathSpaces[i].x = 37; }
        else if (i == 46) { pathSpaces[i].y = 17; pathSpaces[i].x = 31; }
        else if (i == 47) { pathSpaces[i].y = 17; pathSpaces[i].x = 25; }
        else if (i == 48) { pathSpaces[i].y = 17; pathSpaces[i].x = 19; }
        else if (i == 49) { pathSpaces[i].y = 17; pathSpaces[i].x = 13; }
        else if (i == 50) { pathSpaces[i].y = 17; pathSpaces[i].x = 7; }
        else if (i == 51) { pathSpaces[i].y = 17; pathSpaces[i].x = 1; }
}

// initialize the clear spaces' coordinates for each player
for (int i = 0; i < 4; i++) {
        if (i == 0) {
                clearCoords[i][0].x = 7; clearCoords[i][0].y = 15; clearCoords[i][0].position =
52;
                clearCoords[i][1].x = 13; clearCoords[i][1].y = 15; clearCoords[i][1].position =
53;
                clearCoords[i][2].x = 19; clearCoords[i][2].y = 15; clearCoords[i][2].position =
54;
                clearCoords[i][3].x = 25; clearCoords[i][3].y = 15; clearCoords[i][3].position =
55;
                clearCoords[i][4].x = 31; clearCoords[i][4].y = 15; clearCoords[i][4].position =
56;
                clearCoords[i][5].x = 37; clearCoords[i][5].y = 15; clearCoords[i][5].position =
57;
        }
        else if (i == 1) {
                clearCoords[i][0].x = 43; clearCoords[i][0].y = 3; clearCoords[i][0].position =
58;
                clearCoords[i][1].x = 43; clearCoords[i][1].y = 5; clearCoords[i][1].position =
59;
                clearCoords[i][2].x = 43; clearCoords[i][2].y = 7; clearCoords[i][2].position =
60;
                clearCoords[i][3].x = 43; clearCoords[i][3].y = 9; clearCoords[i][3].position =
61;
                clearCoords[i][4].x = 43; clearCoords[i][4].y = 11; clearCoords[i][4].position =
62;
                clearCoords[i][5].x = 43; clearCoords[i][5].y = 13; clearCoords[i][5].position =
63;
        }
        else if (i == 2) {
                clearCoords[i][0].x = 79; clearCoords[i][0].y = 15; clearCoords[i][0].position =
64;
                clearCoords[i][1].x = 73; clearCoords[i][1].y = 15; clearCoords[i][1].position =
65;
                clearCoords[i][2].x = 67; clearCoords[i][2].y = 15; clearCoords[i][2].position =
66;
```

```
                    clearCoords[i][3].x = 61; clearCoords[i][3].y = 15; clearCoords[i][3].position = 
67;
                    clearCoords[i][4].x = 55; clearCoords[i][4].y = 15; clearCoords[i][4].position = 
68;
                    clearCoords[i][5].x = 49; clearCoords[i][5].y = 15; clearCoords[i][5].position = 
69;
                }
                else if (i == 3) {
                    clearCoords[i][0].x = 43; clearCoords[i][0].y = 27; clearCoords[i][0].position = 
70;
                    clearCoords[i][1].x = 43; clearCoords[i][1].y = 25; clearCoords[i][1].position = 
71;
                    clearCoords[i][2].x = 43; clearCoords[i][2].y = 23; clearCoords[i][2].position = 
72;
                    clearCoords[i][3].x = 43; clearCoords[i][3].y = 21; clearCoords[i][3].position = 
73;
                    clearCoords[i][4].x = 43; clearCoords[i][4].y = 19; clearCoords[i][4].position = 
74;
                    clearCoords[i][5].x = 43; clearCoords[i][5].y = 17; clearCoords[i][5].position = 
75;
                }
        }

        // now initialize the path circular linked list for token traversal
        for (int i = 0; i < 52; i++) {
                path.insert(pathSpaces[i]);
        }

}

int getRoll() {
   return rand()%6 + 1;
}
```

```
// check if any block is formed by two same tokens
void checkBlocks() {
	for (int i = 0; i < 4; i++) {
		for (int j = 0; j < numberOfTokens; j++) {
			for (int k = j+1; k < numberOfTokens; k++) {
				if (players[i].tokens[j].position == players[i].tokens[k].position) {
					if (players[i].tokens[j].isFree == true &&
players[i].tokens[k].isFree == true) {
						path.setBlockState(true, i+1,
players[i].tokens[j].position);
					}
				} else {
					if (players[i].tokens[j].position < 52) path.setBlockState(false, 0,
players[i].tokens[j].position);
					if (players[i].tokens[k].position < 52) path.setBlockState(false, 0,
players[i].tokens[k].position);
				}
			}
		}
	}
}

// set token positions
void positions() {
	// a check to see if a token is at home or not
	for (int i = 0, k = 0; i < 4; i++) {
		for (int j = 0; j < numberOfTokens; j++, k++) {
			if (!players[i].tokens[j].isFree) {
				if (!players[i].tokens[j].isClear) {
					players[i].tokens[j].coords.x = homeCoords[k].x;
```

```
                        players[i].tokens[j].coords.y = homeCoords[k].y;
                    } else {
                        players[i].tokens[j].coords.x = clearCoords[i][5].x;
                        players[i].tokens[j].coords.y = clearCoords[i][5].y;
                    }
                }
            }
            k += 4-numberOfTokens;
        }

        // setting all of the tokens for position
        for (int i = 0; i < 4; i++) {
            for (int j = 0; j < numberOfTokens; j++) {
                if (players[i].disqualified == false)
                    boardCoords[players[i].tokens[j].coords.y][players[i].tokens[j].coords.x]
= players[i].tokens[j].display;
                else {
                    for (int k = 0; k < numberOfTokens; k++) {
                        players[i].tokens[k].coords = players[i].startingPoint;
                    }
                    boardCoords[players[i].tokens[j].coords.y][players[i].tokens[j].coords.x]
= ' ';
                }
            }
        }
}

void checkAllTokens();

// check if the token has collided with another opposing token
void crash(int currTokenPosition) {
        for (int i = 0; i < 4; i++) {
                if (i != playerIndex-1) {
                        for (int j = 0; j < numberOfTokens; j++) {
                                if (currTokenPosition == players[i].tokens[j].position) {
                                        if (players[i].tokens[j].isFree == true) {
                                                bool isItSafeSpace = false;
                                                for (int k = 0; k < 8; k++) {
                                                        if (players[i].tokens[j].position ==
safePositions[k]) {
                                                                isItSafeSpace = true;
                                                                break;
                                                        }
                                                }
                                                if (!isItSafeSpace) {
                                                        players[i].tokens[j].isFree = false;
                                                        players[playerIndex-1].hitRecord++;
                                                        players[playerIndex-1].noSixOccurences = 0;
                                                        hitInRound = true;
                                                        someoneHitAToken = true;
                                                        cout << "Hit!\n";
                                                        checkAllTokens();
                                                } else {
                                                        cout << "Space is safe from crashes.\n";
                                                }
                                        }
                                }
                        }
                }
        }
}
```

```
// check if all tokens are free or not; to set the hasFreeToken inside player structure
void checkAllTokens() {
        for (int i = 0; i < 4; i++) {
                bool allLocked = true;
                for (int j = 0; j < numberOfTokens; j++) {
                        if (players[i].tokens[j].isFree == true) {
                                allLocked = false;
                                break;
                        }
                }
                if (allLocked) {
                        players[i].hasFreeToken = false;
                }
        }
}

// special movement function separated from Path structure designed to move player to last steps of clear spaces
void pathToClear(int roll, int position, int token) {
        if (position == 0) { position == 51; }
        int spacesLeft = clearCoords[playerIndex-1][5].position - position;
        cout << spacesLeft << " " << roll << endl;
        if (spacesLeft+1 >= roll) {
                int idx;
                for (int i = 0; i < 6; i++) {
                        if (clearCoords[playerIndex-1][i].position == position) { idx = i; break; }
                }
                while (roll != 0) {
                        idx++;
```

```
            if (players[playerIndex-1].tokens[token].coords.x == clearCoords[playerIndex-1][5].x && players[playerIndex-1].tokens[token].coords.y == clearCoords[playerIndex-1][5].y) break;

            players[playerIndex-1].tokens[token].coords.x = clearCoords[playerIndex-1][idx].x;

            players[playerIndex-1].tokens[token].coords.y = clearCoords[playerIndex-1][idx].y;

            roll--;

        }

        players[playerIndex-1].tokens[token].position = clearCoords[playerIndex-1][idx].position;

    } else {

        cout << "Move is not possible...\n";

    }

}


// main movement function to traverse each piece on board

void movement() {


    char input;

    int turn = 1;

    bool repeat = true;

    /* movement START */

    while (repeat) {


        repeat = false;

        cout << "Player " << playerIndex << ", turn " << turn << ": Number of times no sixes rolled: " << players[playerIndex-1].noSixOccurences << endl;

        cout << "Enter D to roll the dice: ";

        do { cin >> input; } while (input != 'd' && input != 'D');

        int roll, tokenToMove;

        if (input == 'd' || input == 'D') {

            roll = getRoll();
            //if (i==1) { if (roll == 6) roll--; else roll = 6; }
            cout << "Roll obtained: " << roll << endl;
            if (roll == 6) {
                repeat = true;
                players[playerIndex-1].noSixOccurences = 0;
                if (players[playerIndex-1].hasFreeToken == false) {
                    cout << "One token has been released!\n";
                    initialTokens[turn-1] = players[playerIndex-1].tokens[0];
                    players[playerIndex-1].tokens[0].isFree = true;
                    players[playerIndex-1].tokens[0].coords.x =
players[playerIndex-1].startingPoint.x + players[playerIndex-1].tokens[0].slot;
                    players[playerIndex-1].tokens[0].coords.y =
players[playerIndex-1].startingPoint.y;
                    players[playerIndex-1].tokens[0].position = players[playerIndex-
1].startingPosition;
                    players[playerIndex-1].hasFreeToken = true;
                } else {
                    bool allFree = true;
                    for (int j = 0; j < numberOfTokens; j++) {
                        if (players[playerIndex-1].tokens[j].isFree == false &&
players[playerIndex-1].tokens[j].isClear == false) {
                            allFree = false;
                            break;
                        }
                    }
                    if (allFree == false) {
                        int option;
                        cout << "You have a choice!\nEnter 1 to release new
token; enter 2 to move from current tokens: ";
                        do { cin >> option; } while (option != 1 && option !=
2);
                        if (option == 1) {
```

```
int j = 0;
while (players[playerIndex-1].tokens[j].isFree
== true) { j++; }
initialTokens[turn-1] = players[playerIndex-
1].tokens[j];
players[playerIndex-1].tokens[j].isFree = true;
players[playerIndex-1].tokens[j].coords.x =
players[playerIndex-1].startingPoint.x + players[playerIndex-1].tokens[j].slot;
players[playerIndex-1].tokens[j].coords.y =
players[playerIndex-1].startingPoint.y;
players[playerIndex-1].tokens[j].position =
players[playerIndex-1].startingPosition;
} else if (option == 2) {
cout << "Choose a token to move:\n";
for (int j = 0; j < numberOfTokens; j++) {
if (players[playerIndex-
1].tokens[j].isFree == true && players[playerIndex-1].tokens[j].isClear == false) {
cout << "Token " << j+1 <<
endl;
}
}
cout << "Enter: ";
do { cin >> tokenToMove; } while
(players[playerIndex-1].tokens[tokenToMove-1].isFree == false);
initialTokens[turn-1] = players[playerIndex-
1].tokens[tokenToMove-1];
bool insideState = false;
int val = roll;
if (players[playerIndex-1].tokens[tokenToMove-
1].isInside == false) {
bool blockState = false;
Coordinate getCoords =
path.moveToken(val, players[playerIndex-1].tokens[tokenToMove-1].position, players[playerIndex-
1].hitRecord, players[playerIndex-1].startingPosition, playerIndex, blockState, insideState,
tokenToMove-1);
```

```
                                                                                players[playerIndex-
1].tokens[tokenToMove-1].coords.x = getCoords.x + players[playerIndex-1].tokens[tokenToMove-
1].slot;

                                                                                players[playerIndex-
1].tokens[tokenToMove-1].coords.y = getCoords.y;


                                                                                if (val == 0) {
                                                                                        if (!blockState) {
                                                                                                int steps = roll;
                                                                                                while (steps != 0) {

        players[playerIndex-1].tokens[tokenToMove-1].position++;
                                                                                                        if
(players[playerIndex-1].tokens[tokenToMove-1].position == 52)

        players[playerIndex-1].tokens[tokenToMove-1].position = 0;
                                                                                                        steps--;
                                                                                                }
                                                                                        } else {
                                                                                                players[playerIndex-
1].tokens[tokenToMove-1] = initialTokens[turn-1];
                                                                                        }
                                                                                        crash(players[playerIndex-
1].tokens[tokenToMove-1].position);
                                                                                }
                                                                        }
                                                                        if (insideState == true || players[playerIndex-
1].tokens[tokenToMove-1].isInside) {
                                                                                if (players[playerIndex-
1].tokens[tokenToMove-1].isInside == false) {
                                                                                        players[playerIndex-
1].tokens[tokenToMove-1].position = clearCoords[playerIndex-1][0].position;
                                                                                        cout << players[playerIndex-
1].tokens[tokenToMove-1].position << endl;
```

```
                                        }
                                        players[playerIndex-
1].tokens[tokenToMove-1].isInside = true;
                                        pathToClear(val, players[playerIndex-
1].tokens[tokenToMove-1].position, tokenToMove-1);
                                    }
                                }
                            } else {
                                cout << "Choose a token to move:\n";
                                for (int j = 0; j < numberOfTokens; j++) {
                                    if (players[playerIndex-1].tokens[j].isFree ==
true && players[playerIndex-1].tokens[j].isClear == false) {
                                        cout << "Token " << j+1 << endl;
                                    }
                                }
                                cout << "Enter: ";
                                do { cin >> tokenToMove; } while (players[playerIndex-
1].tokens[tokenToMove-1].isFree == false);
                                initialTokens[turn-1] = players[playerIndex-
1].tokens[tokenToMove-1];
                                bool insideState = false;
                                int val = roll;
                                if (players[playerIndex-1].tokens[tokenToMove-
1].isInside == false) {
                                    bool blockState = false;
                                    Coordinate getCoords = path.moveToken(val,
players[playerIndex-1].tokens[tokenToMove-1].position, players[playerIndex-1].hitRecord,
players[playerIndex-1].startingPosition, playerIndex, blockState, insideState, tokenToMove-1);
                                    players[playerIndex-1].tokens[tokenToMove-
1].coords.x = getCoords.x + players[playerIndex-1].tokens[tokenToMove-1].slot;
                                    players[playerIndex-1].tokens[tokenToMove-
1].coords.y = getCoords.y;

                                    if (val == 0) {
                                        if (!blockState) {
                                            int steps = roll;
                                            while (steps != 0) {
                                                players[playerIndex-
1].tokens[tokenToMove-1].position++;
                                                if (players[playerIndex-
1].tokens[tokenToMove-1].position == 52)

    players[playerIndex-1].tokens[tokenToMove-1].position = 0;
                                                steps--;
                                            }
                                        } else {
                                            players[playerIndex-
1].tokens[tokenToMove-1] = initialTokens[turn-1];
                                        }
                                        crash(players[playerIndex-
1].tokens[tokenToMove-1].position);
                                    }
                                }
                                if (insideState == true || players[playerIndex-
1].tokens[tokenToMove-1].isInside)  {
                                    if (players[playerIndex-1].tokens[tokenToMove-
1].isInside == false) {
                                        players[playerIndex-
1].tokens[tokenToMove-1].position = clearCoords[playerIndex-1][0].position;
                                    }
                                    players[playerIndex-1].tokens[tokenToMove-
1].isInside = true;
                                    pathToClear(val, players[playerIndex-
1].tokens[tokenToMove-1].position, tokenToMove-1);
                                }
                                //players[i-1].tokens[tokenToMove-1].position =
getPos(players[i-1].tokens[tokenToMove-1].coords);
                                //sleep(2);
```

```
                                        }
                                }
                        } else {
                                if (players[playerIndex-1].hasFreeToken == false) {
                                        cout << "Cannot play; passing turn to next player.\n";
                                } else {
                                        cout << "Choose a token to move:\n";
                                        for (int j = 0; j < numberOfTokens; j++) {
                                                if (players[playerIndex-1].tokens[j].isFree == true &&
players[playerIndex-1].tokens[j].isClear == false) {
                                                        cout << "Token " << j+1 << endl;
                                                }
                                        }
                                        cout << "Enter: ";
                                        do {
                                                tokenToMove = -1;
                                                if (tokenToMove < 1 || tokenToMove > 4) { cin >>
tokenToMove; }
                                        } while (players[playerIndex-1].tokens[tokenToMove-1].isFree
== false);
                                        initialTokens[turn-1] = players[playerIndex-
1].tokens[tokenToMove-1];
                                        bool insideState = false;
                                        int val = roll;
                                        if (players[playerIndex-1].tokens[tokenToMove-1].isInside ==
false) {
                                                bool blockState = false;
                                                Coordinate getCoords = path.moveToken(val,
players[playerIndex-1].tokens[tokenToMove-1].position, players[playerIndex-1].hitRecord,
players[playerIndex-1].startingPosition, playerIndex, blockState, insideState, tokenToMove-1);
                                                players[playerIndex-1].tokens[tokenToMove-1].coords.x
= getCoords.x + players[playerIndex-1].tokens[tokenToMove-1].slot;
```

```
                                                players[playerIndex-1].tokens[tokenToMove-1].coords.y
= getCoords.y;

                                                if (val == 0) {
                                                        if (!blockState) {
                                                                int steps = roll;
                                                                while (steps != 0) {
                                                                        players[playerIndex-
1].tokens[tokenToMove-1].position++;
                                                                        if (players[playerIndex-
1].tokens[tokenToMove-1].position == 52)
                                                                                players[playerIndex-
1].tokens[tokenToMove-1].position = 0;
                                                                        steps--;
                                                                }
                                                        } else {
                                                                players[playerIndex-
1].tokens[tokenToMove-1] = initialTokens[turn-1];
                                                        }
                                                        crash(players[playerIndex-
1].tokens[tokenToMove-1].position);
                                                }
                                        }
                                        if (insideState == true || players[playerIndex-
1].tokens[tokenToMove-1].isInside) {
                                                if (players[playerIndex-1].tokens[tokenToMove-
1].isInside == false) {
                                                        players[playerIndex-1].tokens[tokenToMove-
1].position = clearCoords[playerIndex-1][0].position;
                                                }
                                                players[playerIndex-1].tokens[tokenToMove-1].isInside
= true;
                                                pathToClear(val, players[playerIndex-
1].tokens[tokenToMove-1].position, tokenToMove-1);
```

```
                                        }
                                }
                        }

                }
                turn++;
                if (roll != 6 && someoneHitAToken == false) {
                        players[playerIndex-1].noSixOccurences++;
                        turn = 1;
                        repeat = false;
                } else {
                        players[playerIndex-1].noSixOccurences = 0;
                        someoneHitAToken = false;
                }
                if (turn >= 4) {
                        cout << "3 sixes rolled! Skipping to next player... "; // skip to next player
                        turn = 1;
                        repeat = false;
                }

                for (int j = 0; j < numberOfTokens; j++) { // check if any token is at ending position
                        if (players[playerIndex-1].tokens[j].position == clearCoords[playerIndex-
1][5].position) {
                                cout << "Token " << players[playerIndex-1].tokens[j].slot+1 << " of
player " << playerIndex << " has reached home!\n";
                                players[playerIndex-1].tokens[j].isClear = true;
                                players[playerIndex-1].tokens[j].isFree = false;
                        }
                }

        }
        /* movement END */

}

//37-40 / 43-46 / 49-52 (spanning from 1 to 11 and 19 to 29 (by increment of 2))
//1-4 / 7-10 / 13-16 / 19-22 / 25-28 / 31-34 (spanning from 13-17 (by increment of 2))
//55-58 / 61-64 / 67-70 / 73-76 / 79-82 / 85-88 (spanning from 13-17 (by increment of 2))

// draw the board after adjusting for blocks, crashes, and general positions of each and every token
void drawBoard() {
        // compile the board into the string
        board = "";
  board += ".........................................................................................";
        board += ".                    .   .   .   .                    .";
        board += ".                    ......+-----/*****\\                          .";
        board += ".                    .   |   |   | <--                 .";
        board += ".                    /*****\\-----\\*****/                          .";
        board += ".                    |   |   |   .                    .";
        board += ".       .  .  .  .   \\*****/-----|......       .  .  .  .         .";
        board += ".                    .   |   |   .                    .";
        board += ".                    ......|-----|......                 .";
        board += ".                    .   |   |   .                    .";
        board += ".      |             ......|-----|......                 .";
        board += ".      V               .   |   |   .                  .";
        board += "....../*****\\......................+----------------+................/*****\\............";
        board += ".   |   |   .   .   .   |            |   .   .   |   |   .   .";
        board += "......\\*****/----------------------+            +----------------\\*****/-----+......";
        board += ".   |   |   |   |   |   |   CLEAR   |   |   |   |   |   |   .";
        board += "......+-----/*****\\----------------+            +--------------------/*****\\......";
        board += ".   .   |   |   .   .   |            |   .   .   .   |   |   .";
        board += "............\\*****/.................+-----------------+.......................\\*****/......";
        board += ".                              .    |    |    .                       ^       .";
        board += ".                              ......|-----|......                       |       .";
        board += ".                              .    |    |    .                              .";
        board += ".                              ......|-----|......                             .";
        board += ".                              .    |    |    .                              .";
        board += ".          .  .  .  .          ......|-----/*****\\          .  .  .  .          .";
        board += ".                              .    |    |    |                              .";
        board += ".                              /*****\\-----\\*****/                              .";
        board += ".                          --> |    |    |    .                              .";
        board += ".                              \\*****/-----+......                              .";
        board += ".                              .    .    .    .                              .";
        board += "..........................................................................................";
        // transfer it over to the coordinates list
        for (int i = 0, k = 0; i < 31; i++) {
                for (int j = 0; j < 91; j++, k++) {
                        boardCoords[i][j] = board[k];
                }
        }
        // call function for checking block states of each position player is on
        checkBlocks();

        // this is where the tokens will have their positions set before the displaying of the board
        positions();

        for (int i = 0; i < 31; i++) {
                for (int j = 0; j < 91; j++) {
                        cout << boardCoords[i][j];
                }
                cout << endl;
        }
}

void* playerThread(void* arg) {
        int oldState, oldType;
        pthread_setcancelstate(PTHREAD_CANCEL_ENABLE, &oldState);
        pthread_setcanceltype(PTHREAD_CANCEL_ASYNCHRONOUS, &oldType);
        while (1) {

                bool allClear = true;
                for (int j = 0; j < numberOfTokens; j++) {
                        if (players[playerIndex-1].tokens[j].isClear == false) {
                                allClear = false;
                                break;
                        }
                }
                if (allClear == true) {
                        players[playerIndex-1].hasWon = true;
                        whoWon = playerIndex;
                        someoneWonArr[playerIndex-1] = true;
                        someoneWon = true;
                }

                while (hasCompletedTask[playerIndex-1] || players[playerIndex-1].disqualified == true ||
someoneWonArr[playerIndex-1] == true) {
                        playerIndex++;
                        if (playerIndex > 4) playerIndex = 1;
                }

                sem_wait(&semaphore);
                movement();
                sem_post(&semaphore);

                sem_wait(&semaphore);
                drawBoard();
                hasCompletedTask[playerIndex-1] = 1;
                sem_post(&semaphore);
        }
        pthread_exit(NULL);
}

void* masterThreadFunc(void* arg) {

        void* status[4];

        for (int i = 0; i < 4; i++) {
                pthread_create(&playerThreads[i], NULL, playerThread, NULL);
        }

        while (1) {

                if (closeGame == true)
                        break;

                if (players[playerIndex-1].disqualified == true) {
                        players[playerIndex-1].winState = 2; // means it lost
                        hasCompletedTask[playerIndex-1] = 2;
                        pthread_cancel(playerThreads[playerIndex]);
                        cout << "Cancelled thread\n";
                }
```

```
            bool someoneStillLeft = false;
            for (int i = 0; i < 4; i++) {
                if (someoneWonArr[i] == false) { someoneStillLeft = true; break; }
            }

            if (someoneStillLeft == false) {
                cout << "All players have finished the game.\n";
                break;
            }

            if (someoneWon) {
                cout << "\nPlayer---" << whoWon << "---has won the game!\n";
                pthread_cancel(playerThreads[whoWon-1]);
                someoneWon = false;
            }

            if (hasCompletedTask[0] && hasCompletedTask[1] && hasCompletedTask[2] &&
hasCompletedTask[3]) {
                if (hasCompletedTask[0] == 1) hasCompletedTask[0] = 0;
                if (hasCompletedTask[1] == 1) hasCompletedTask[1] = 0;
                if (hasCompletedTask[2] == 1) hasCompletedTask[2] = 0;
                if (hasCompletedTask[3] == 1) hasCompletedTask[3] = 0;
                cout << "---\nRound End\n---\n";
                if (hitInRound) {
                    cout << "Hit records:\n";
                    for (int j = 0; j < 4; j++) {
                        cout << "Player " << j+1 << " hit count: " <<
players[j].hitRecord << endl;
                    }
                    hitInRound = false;
                    }
                }
            }

            pthread_exit((void*)true);
}


int main() {
            init();
            sem_init(&semaphore, 0, 1);
   srand(time(NULL));
            cout << "Enter the number of tokens for every player: ";
            do {
                        cin >> numberOfTokens; // "input will be take only once and assigned to each player"
            } while (numberOfTokens < 1 || numberOfTokens > 4);
            pthread_t playerThreads[4];
   players = new Player[4];
            // initialize all of the tokens' numbers, slots, starting points and positions
            for (int i = 0; i < 4; i++) {
                        players[i].numberOfTokens = numberOfTokens;
                        players[i].tokens = new Token[players[i].numberOfTokens];
                        for (int j = 0; j < numberOfTokens; j++) {
                                    players[i].tokens[j].display = ('1' + i);
                                    players[i].tokens[j].slot = j;
                        }
                        if (i == 0) { players[i].startingPoint.x = 7; players[i].startingPoint.y = 13;
players[i].startingPosition = 2; }
                        else if (i == 1) { players[i].startingPoint.x = 49; players[i].startingPoint.y = 3;
players[i].startingPosition = 15; }
                        else if (i == 2) { players[i].startingPoint.x = 37; players[i].startingPoint.y = 27;
players[i].startingPosition = 41; }
```

```
			else if (i == 3) { players[i].startingPoint.x = 79; players[i].startingPoint.y = 17;
players[i].startingPosition = 28; }
		}

		void* status;

		pthread_create(&masterThread, NULL, masterThreadFunc, NULL);
		pthread_join(masterThread, &status);

		while (1) {
			bool* stateOfGame = (bool*) status;
			if (*stateOfGame) break;
		}
		// display players' hits information at the end of game
		cout << "Players' hit records:\n";
		for (int i = 0; i < 4; i++) {
			cout << "Player " << i+1 << " hit count: " << players[i].hitRecord << endl;
		}

   pthread_exit(NULL);
}
```

System Specifications:

This project was worked on two systems simultaneously:

- System 1:
  - Core i5 4200U (4th gen), 2 CPUs - 4 Threads
  - 8.00 Gigabytes of RAM (DDR3)
  - 240 GBs Solid State Drive
- System 2:
  - Core i5 6200U (6th gen), 2 CPUs - 4 Threads
  - 8.00 Gigabytes of RAM (DDR3)
  - 1 TB Hard Disk Drive + 120 GBs Solid State Drive

Implement this concept in other scenario:

Suppose that a racing competition is being held between 6 players at a time. Each player will have a thread to manage their average speed (at each second) which can displace a few meters/second in speed. A master thread in the form of the judge will watch and determine which person (in programming's case, which thread) has reached the finish line first, declaring said person (thread) as the winner of the race.

Programmatically challenging implementations and other features:

- Instead of the usual array-based implementation for Ludo board's spaces, a circular linked list was used for the outer loop with conditions (when a token reaches said position and the player has crashed at least one opposing token). The inner loop was based on normal array coordinates

- Structures were heavily used to implement most of the variables, including coordinates – which in turn was used to design the board and its set of coordinates the tokens could traverse on.

- Instead of running the simulation, input is being asked from user to validate which token they would like to move if given the choice.

- The board has been visually designed to notify where the safe spaces are and what starting points each player has (areas marked with asterisks and arrows respectively)

Contribution of each person in project:

- Project code: **i190599** & **i190695**
- Phase 1 & 2 pseudo codes: **- i190599**
- Concepts of OS used: **i190599** & **i190695**
- System specifications: **i190599** & **i190695**
- Implement concept in other scenario: **i190695**
- Project report: **i190599** & **i190695**