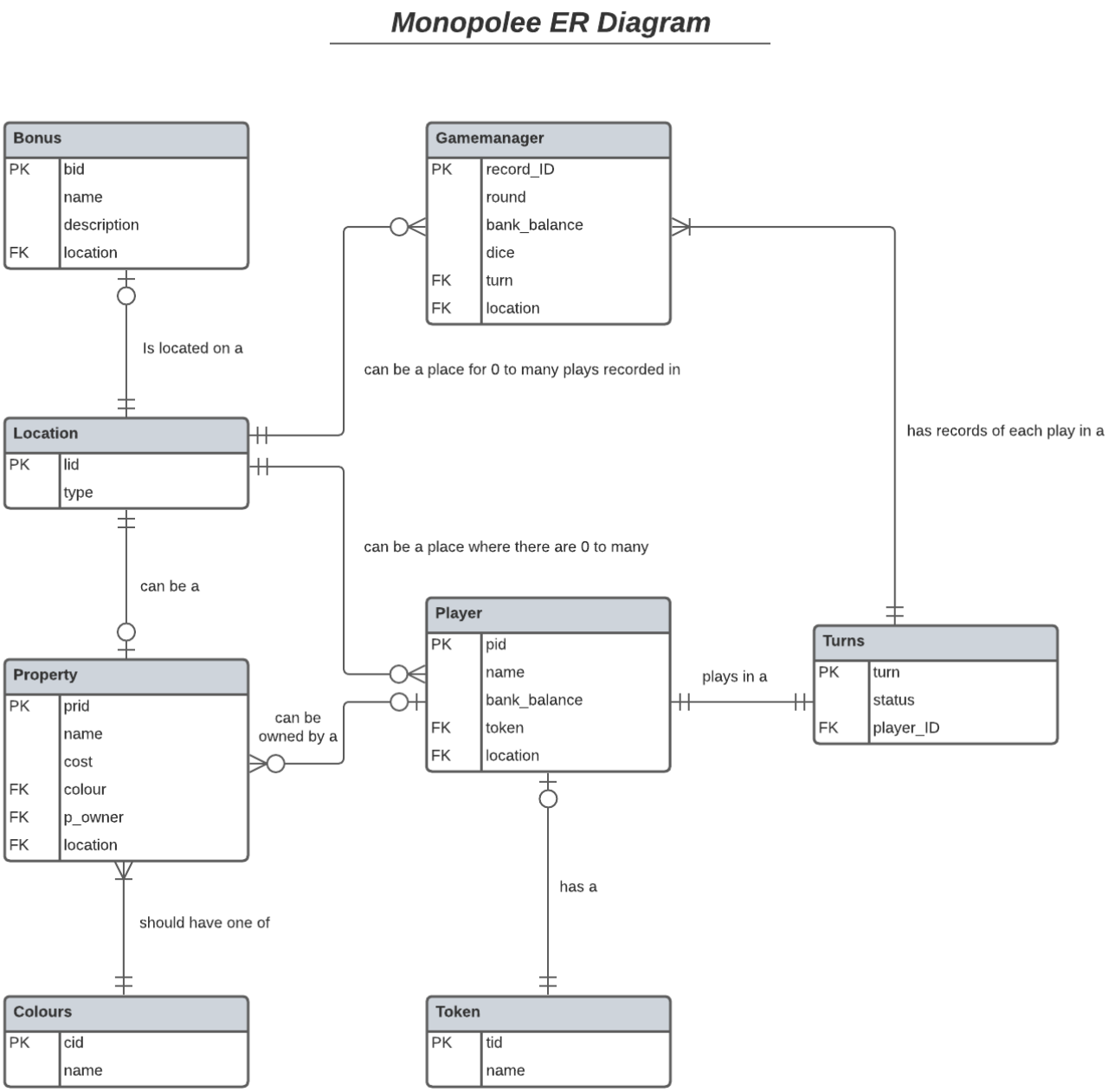
DATA70141 Assignment 1 – Monopolee

1. Entity Relationship Diagram

Figure 1. Monopolee ER Diagram



Soomin Myung, 11155827

2. Design choices

a. Each table will have its own Primary Key in INT, even though there is a “UNIQUE” VARCHAR column in it. It is because using Primary Keys in VARCHAR string names can make the game vulnerable to any errors by typos.

b. The “Gamemanager” table has “dice” column to get dice numbers for each turn by INSERT queries. The table also has “round”, “bank_balance”, “turn”, and “location” columns along with the Primary Key “record_ID”, to record the gameplay as an audit table. Each record (row) gets inserted to the table with one “location” number and one “turn” number.
Each “turn” number from “Turns” must appear in “Gamemanager” at least once because each player will play at least once in the first round. Meanwhile, each “location” number can appear 0 to many times. The “bank_balance” is the bank balance status of a player at a particular turn. It is not a Foreign Key to the “Player” table because it is not UNIQUE.
The columns “turn”, “location”, and “bank_balance” do not imply redundancy; These columns are to record the gameplay by INSERT queries, while the ones in other tables like “Turns” and “Players” are to reflect the recent status of the game by UPDATE queries.

c. The “Location” table with the “lid” Primary Key is for the 16 tiles of the game board, each of which can be either a bonus or a property. The “type” column is set to 1 for property tiles and 0 for bonus tiles.

d. The “Bonus” table is for the bonus tiles with the “name”, “description”, and the Primary Key “bid”. It can be placed on a tile with a “location” number, which is a Foreign Key to the “Location” table.

e. Each property can be on one of the tiles; therefore, the “Property” table has also “location” column like the “Bonus” table. The “cost” is the money for purchasing the property or for the rent fee if the property is owned by someone else. Each of the properties should have one of the colours from the “Colours” table by the “colour” Foreign Key. The “p_owner” Foreign Key to the “Player” table is set to a certain player’s ID when he buys the property.

f. Each player chooses a token to play the game by “token” Foreign Key to the “Token” table. The “location” Foreign Key to the “Location” table is the location of the player; a player should be on a location, and a location can have 0 to many players at a certain point in time.

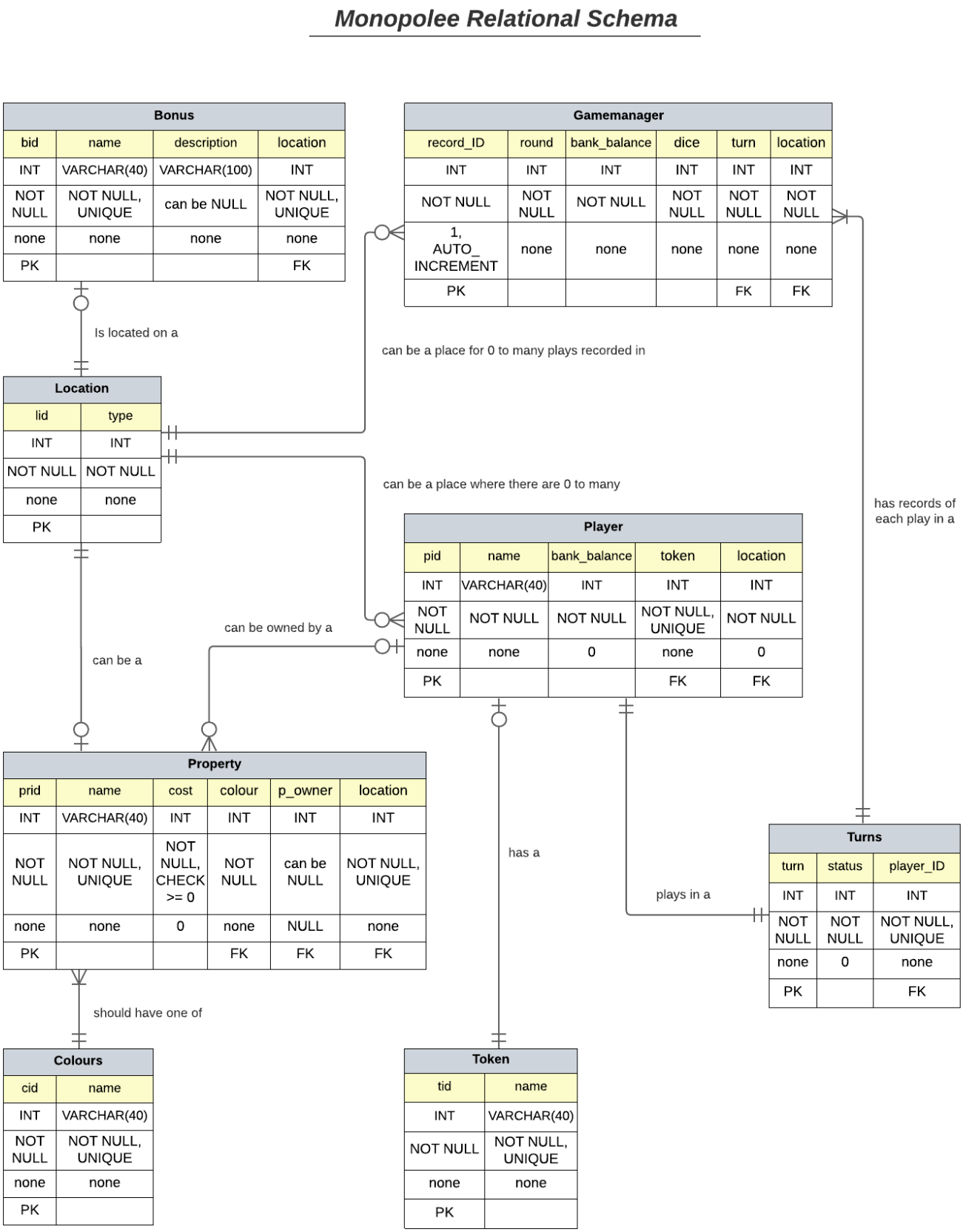
g. The “Colours” table was made separately to prevent any property from using a colour not specified in the rule.

h. The “Turns” table is to manage the turns separately. When a certain player goes bankrupt and can no longer play the game, the “status” becomes 2, and he will not get his turn anymore.

3. Relational Database Schema and Initial State of the Game

3.1. Relational Database Schema

Figure 2. Monopolee Relational Schema



3.2. Mapping choices

a. In “bonus” table, “location” is set UNIQUE to avoid any situation when more than one bonus is on the same tile. “description” can be NULL because it is not necessary for the game flow.

b. In “Property” table, the default “cost” is 0, and the values cannot be negative. “p_owner” can be NULL until it is set to a certain player’s ID as the owner. “location” is UNIQUE to prevent any more than one property is on the same tile.

c. The “Turns”, “Colours”, and “Token” tables are the result of normalisation; “Turns” and “Token” have been separated from the “Player” table because the columns of the tables are not dependent on the Primary Key of the “Player” table “pid”. “Colours” was separated from the “Property” table for the same reason. It also prevents players from getting a token or turn not specified in the rule and properties from being in a colour not from the rule.

d. The “Location”, “Bonus”, and “Property” tables could be in the same table, but they have been separated into 3 tables by normalisation considering that the variables do not depend on the “lid” Primary Key and redundancies.

e. The number of players gets controlled by a BEFORE INSERT TRIGGER “add_player” on the “Player” table (Figure 3). And there is no restriction on the number of rows in “Token” table because it is a matter of choice as long as we control the number of players.

f. The 4 triggers (b_location_chk_1, b_location_chk_2, pr_location_chk_1, pr_location_chk_2) in Figure 3 are to prevent any bonus or property from using a location for the other type.

3.3. Initial State of the Game

Regarding the given initial state of the game, the costs (rent fees) of “Oak House” and “Owens Park” doubled in advance when inserted into the “Property” table. Meanwhile, starting from the given initial locations do not cause any landing effect of the locations.

Figure 3. Queries to set the initial state of the game

```
CREATE TABLE Location (
    lid INT NOT NULL PRIMARY KEY,
    type INT NOT NULL #0: bonus, 1: property
);

CREATE TABLE Bonus (
    bid INT NOT NULL PRIMARY KEY,
    name VARCHAR(40) NOT NULL UNIQUE, #Unique bonus name
    description VARCHAR(100),
    location INT UNIQUE NOT NULL, #To prevent any duplicate using the same location
    CONSTRAINT FK_b_location FOREIGN KEY (location)
```

```
        REFERENCES Location (lid)
);

CREATE TABLE Token (
    tid INT NOT NULL PRIMARY KEY,
    name VARCHAR(40) NOT NULL UNIQUE #To prevent any duplicate using the same name
);

CREATE TABLE Player (
    pid INT NOT NULL PRIMARY KEY,
    name VARCHAR(40) NOT NULL,
    token INT NOT NULL UNIQUE,
    bank_balance INT NOT NULL DEFAULT 0,
    location INT NOT NULL DEFAULT 0,
    CONSTRAINT FK_token FOREIGN KEY (token)
        REFERENCES Token (tid),
    CONSTRAINT FK_location FOREIGN KEY (location)
        REFERENCES Location (lid)
);

#Make a separate Colour table to prevent any property from using a colour which is not from the rule
CREATE TABLE Colours (
    cid INT NOT NULL PRIMARY KEY,
    name VARCHAR(40) NOT NULL UNIQUE #Colour names should be unique
);

CREATE TABLE Property (
    prid INT NOT NULL PRIMARY KEY,
    name VARCHAR(40) NOT NULL UNIQUE, #Unique property name
    cost INT NOT NULL DEFAULT 0 CHECK (cost >= 0), #Cost can't be negative
    colour INT NOT NULL,
    p_owner INT DEFAULT NULL,
    location INT NOT NULL UNIQUE, #UNIQUE to prevent any duplicate using the same location
    CONSTRAINT FK_colour FOREIGN KEY (colour)
        REFERENCES Colours (cid),
    CONSTRAINT FK_p_owner FOREIGN KEY (p_owner)
        REFERENCES Player (pid),
    CONSTRAINT FK_p_location FOREIGN KEY (location)
        REFERENCES Location (lid)
);

CREATE TABLE Turns (
    turn INT NOT NULL,
    #UNIQUE to prevent any player from being designated for more than 1 turn
    player_ID INT NOT NULL UNIQUE,
    #status = 0: playing, 1: IN JAIL, 2: defeated, 3: winner
    status INT NOT NULL DEFAULT 0,
    CONSTRAINT FK_t_player FOREIGN KEY (player_ID)
        REFERENCES Player (pid),
    PRIMARY KEY (turn , player_ID)
);

#Gamemanager table which gets the dice number and controls the game flow
#It is also to record the track of the gameplay as an audit table at the same time
```

```
CREATE TABLE Gamemanager (
    record_ID INT PRIMARY KEY AUTO_INCREMENT,
    round INT NOT NULL, #round adds up when every turn is played
    turn INT NOT NULL, #1 turn for 1 player
    dice INT NOT NULL, #dice number as an input
    location INT NOT NULL, #to track the location everytime a certain player moves and lands on it
    #to track the bank balance of a certain player after he plays his turn
    bank_balance INT NOT NULL,
    CONSTRAINT FK_turn FOREIGN KEY (turn)
        REFERENCES Turns (turn),
    CONSTRAINT FK_g_location FOREIGN KEY (location)
        REFERENCES Location (lid)
);

#gameView to show the status of the game at certain time
CREATE VIEW gameView AS
    SELECT (SELECT MAX(round) FROM Gamemanager) AS "Round", t.player_ID AS "Player No.",
        p.name AS "Player name", p.location AS "Location",
        group_concat(DISTINCT pr.name) AS "Properties owned", p.bank_balance AS "Bank balance",
        (CASE t.status WHEN 0 THEN "Playing" WHEN 1 THEN "IN JAIL" WHEN 2 THEN "Defeated"
        WHEN 3 THEN "Winner!" END) AS "Status"
    FROM Turns AS t
        INNER JOIN Player AS p ON t.player_ID = p.pid
        INNER JOIN GameManager AS gm ON gm.turn = t.turn
        LEFT JOIN Property AS pr ON t.player_ID = pr.p_owner
    GROUP BY t.turn;

DELIMITER $$

#Trigger to prevent more than 6 players from being added to the game
CREATE TRIGGER add_player
BEFORE INSERT ON Player FOR EACH ROW
BEGIN
    DECLARE cnt_player INT;
    SELECT
        COUNT(*)
    INTO cnt_player FROM Player;
    IF cnt_player >= 6 THEN
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'UP to 6 players can be registered per game.';
    END IF;
END$$

#Trigger to prevent bonus tiles from being located on a location for properties
CREATE TRIGGER b_location_chk_1
BEFORE INSERT ON Bonus FOR EACH ROW
BEGIN
        DECLARE type INT;
        SELECT Location.type INTO type FROM Location WHERE lid = NEW.location;
        IF type = 1 THEN
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Please check the location type';
        END IF;
    END$$

#Trigger to prevent bonus tiles from being located on a location for properties
```

```
    CREATE TRIGGER b_location_chk_2
    BEFORE UPDATE ON Bonus FOR EACH ROW
    BEGIN
        DECLARE type INT;
        SELECT Location.type INTO type FROM Location WHERE lid = NEW.location;
        IF type = 1 THEN
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Please check the location type';
        END IF;
    END$$

#Trigger to prevent property tiles from being located on a location for bonuses
    CREATE TRIGGER pr_location_chk_1
    BEFORE INSERT ON Property FOR EACH ROW
    BEGIN
        DECLARE type INT;
        SELECT Location.type INTO type FROM Location WHERE lid = NEW.location;
        IF type = 0 THEN
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Please check the location type';
        END IF;
    END$$

#Trigger to prevent property tiles from being located on a location for bonuses
    CREATE TRIGGER pr_location_chk_2
    BEFORE UPDATE ON Property FOR EACH ROW
    BEGIN
        DECLARE type INT;
        SELECT Location.type INTO type FROM Location WHERE lid = NEW.location;
        IF type = 0 THEN
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Please check the location type';
        END IF;
    END$$

DELIMITER ;

#Insert values to set the game
INSERT INTO Location  #(lid, type)
    VALUES (0, 0),
    (1, 1),
    (2, 0),
    (3, 1),
    (4, 0),
    (5, 1),
    (6, 0),
    (7, 1),
    (8, 0),
    (9, 1),
    (10, 0),
    (11, 1),
    (12, 0),
    (13, 1),
    (14, 0),
    (15, 1);

INSERT INTO Colours  #(cid, name)
```

```
    VALUES (0, "Yellow"),
    (1, "Green"),
    (2, "Orange"),
    (3, "Blue");

INSERT INTO Token  #(tid, name)
    VALUES (0, "Dog"),
    (1, "Car"),
    (2, "Battle-ship"),
    (3, "Top hat"),
    (4, "Thimble"),
    (5, "Boot");

INSERT INTO Player  #(pid, name, token, bank_balance, location)
    VALUES (0, "Jane", 1, 150, 13),
    (1, "Norman", 4, 250, 1),
    (2, "Mary", 2, 190, 8),
    (3, "Bill", 0, 500, 11);

INSERT INTO Property (prid, name, cost, colour, p_owner, location)
    VALUES     (0, "Kilburn", 120, 0, NULL, 1),
    (1, "Uni Place", 100, 0, 2, 3),
    (2, "Victoria", 75, 1, 3, 5),
    (3, "Piccadilly", 35, 1, NULL, 7),
    (4, "Oak House", 200, 2, 1, 9), #Original cost = 100
    (5, "Owens Park", 60, 2, 1, 11), #Original cost = 30
    (6, "AMBS", 400, 3, NULL, 13),
    (7, "Co-Op", 30, 3, 0, 15);

INSERT INTO Bonus  #(bid, name, description, location)
    VALUES (0, "GO", "Collect 200 pounds", 0),
    (1, "Chance 1", "Pay each of the other players 50 pounds", 2),
    (2, "IN JAIL", "must roll a 6 to get out", 4),
    (3, "Community Chest 1", "For winning a Beauty Contest, you win 100 pounds", 6),
    (4, "Free Parking", "No action", 8),
    (5, "Chance 2", "Move forward 3 spaces", 10),
    (6, "Go to Jail", "Go to Jail, do not pass GO, do not collect 200 pounds", 12),
    (7, "Community Chest 2", "Your library books are overdue. Play a fine of 30 pounds", 14);

INSERT INTO Turns  #(turn, player_ID, status)
    VALUES (0, 0, 0),
    (1, 1, 0),
    (2, 2, 0),
    (3, 3, 0);

#Session variable, total number of locations (tiles)
SET @no_of_tiles = (SELECT COUNT(*) FROM Location);
#Session variable, total number of active players of initial state
SET @init_no_of_players = (SELECT COUNT(*) FROM Player);
SET @turn = 0; #Session variable for turn number, turn starts from 0
#Session variable for the location of "IN JAIL" tile, now it is designated to location 4
SET @jail = (SELECT location FROM Bonus WHERE name = "IN JAIL");
SET @round = 1; #Session variable for round number, round starts from 1
```

4. Gameplay and gameView

4.1. Game design choices

a. When a player is “IN JAIL”, he has to roll a 6 to escape (**R3**), but it does not actually move the player 6 tiles. He has to roll again immediately to move from that “IN JAIL” tile.

b. When a player lands on “IN JAIL” tile, he will be imprisoned; therefore, he has to roll a 6 to get out.

c. When a certain player can’t pay any rent fee or money for bonuses, he goes bankrupt and defeated regardless of how many properties he has. He also immediately loses all property ownerships. This is to make the game more thrilling.

d. Even if a player lands on a property owned by another and is unable to pay the rent fee, the owner receives the full rent fee. The same goes for the bonus “Chance 1”. The other players (still playing and undefeated) will receive 50 pounds regardless of the situation.

e. When someone monopolises a particular colour, it doubles the rent fees of the properties in that colour (**R2**). Even after the owner goes defeated and loses all the ownerships, the rent fees remain double. And the rent fees can even double once again when another player monopolises that colour again. This is to make the game more thrilling and also reflects the reality that overheated real estate prices do not fall easily.

f. If all but one player gets defeated, the last player standing becomes the winner of the game. The game is over and no one can move by rolling a dice.

4.2. How it works

a. This game is operated by 1 trigger, 7 procedures, and 1 function (The other 5 triggers are to constraint the number of players or for database integrity). When a dice number gets inserted into “Gamemanager”, the before insert trigger “dice_trigger” gets activated and controls the game flow by using the procedures, functions, and several session variables such as “@turn”, “@round”, “@no_of_tiles”, “@init_no_of_tiles”, and “@jail”.

b. The “dice_trigger” calls the “move_player” procedure, which updates “Player” to move the player in a manner that conforms to **R3**. It adds up the player’s bank balance when he goes through the “GO” tile (**R4**). If the dice number is not 6, then this procedure calls the “player_landing” procedure to activate landing effects (**R5**).

c. The “player_landing” procedure is to check the location type a player landed on and call “landing_on_property” or “landing_on_bonus” according to the location type.

d. The “landing_on_property” is called when a player lands on a property. It checks the property’s ownership. If it is owned by someone else then the player has to pay the rent fee by the function “change_balance” with the “isBuyingProperty” FALSE. Or else, he can buy the property by “change_balance” with the “isBuyingProperty” TRUE (**R1**, **R2**).

e. The “change_balance” function is to make changes in the players’ bank balance. When “isBuyingProperty” is FALSE and a certain player is unable to pay, then he goes bankrupt and defeated, losing all property ownerships. The payment will be made to the recipients regardless of the situation. It then calls “winner_checker” procedure to see if there is only last player standing as the winner.

f. When someone buys a property, the “monopoly_checker” gets called to see if he monopolises the colour. If so, it will update “Property” to double the rent fees in that colour (**R2**).

g. The “landing_on_bonus” procedure to activate the bonus effects accordingly (**R6**, **R7**) by using case statement. In Figure 4, procedure can also call the “change_balance” procedure, and it may lead to a player going bankrupt in some cases. However, this mechanism is not shown in Figure 4 in detail because of the lack of the space on the paper.

h. The “select_next_player” procedure is to change the session variables “@turn” and “@round” when a turn is finished, according to the current situation (**R3**, **R5**, **R6**).

i. Figure 4 is to show how the game works by a flow chart, and Figure 5 is the actual MySQL code.

Figure 4. Monopolee flow chart

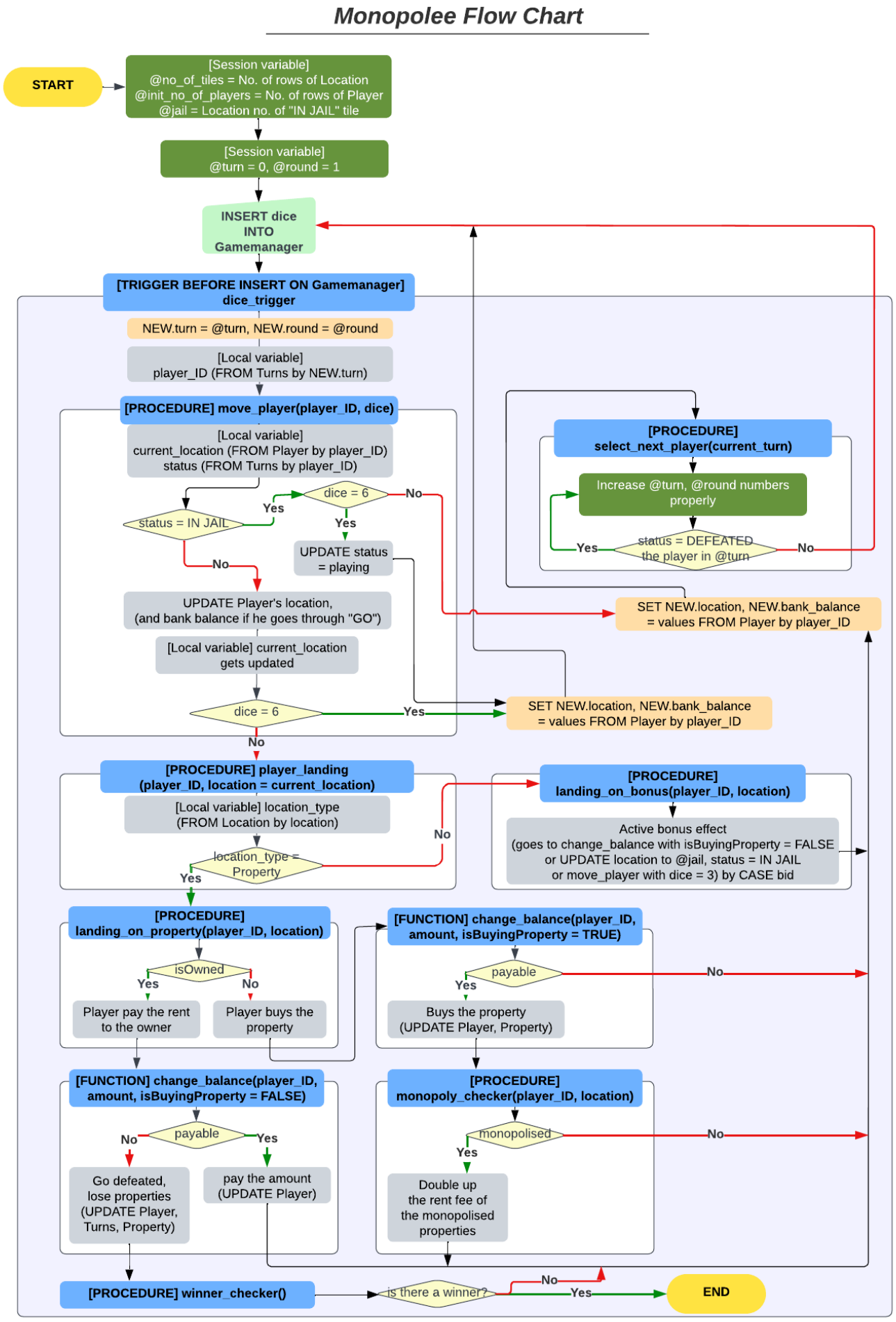


Figure 5. The code for Monopolee

```
DROP TABLE IF EXISTS Property;
DROP TABLE IF EXISTS Colours;
DROP TABLE IF EXISTS Gamemanager;
DROP TABLE IF EXISTS Turns;
DROP TABLE IF EXISTS Player, Token, Bonus, Location;
DROP PROCEDURE IF EXISTS select_next_player;
DROP PROCEDURE IF EXISTS player_landing;
DROP PROCEDURE IF EXISTS landing_on_property;
DROP PROCEDURE IF EXISTS landing_on_bonus;
DROP PROCEDURE IF EXISTS monopoly_checker;
DROP PROCEDURE IF EXISTS move_player;
DROP PROCEDURE IF EXISTS winner_checker;
DROP FUNCTION IF EXISTS change_balance;
DROP VIEW IF EXISTS gameView;

SET max_sp_recursion_depth = 10; #To allow some recursions in the "move_player" procedure

CREATE TABLE Location (
    lid INT NOT NULL PRIMARY KEY,
    type INT NOT NULL #0: bonus, 1: property
);

CREATE TABLE Bonus (
    bid INT NOT NULL PRIMARY KEY,
    name VARCHAR(40) NOT NULL UNIQUE, #Unique bonus name
    description VARCHAR(100),
    location INT UNIQUE NOT NULL, #To prevent any duplicate using the same location
    CONSTRAINT FK_b_location FOREIGN KEY (location)
        REFERENCES Location (lid)
);

CREATE TABLE Token (
    tid INT NOT NULL PRIMARY KEY,
    name VARCHAR(40) NOT NULL UNIQUE #To prevent any duplicate using the same name
);

#Make a separate Colour table to prevent any property from using a colour which is not from the rule
CREATE TABLE Colours (
    cid INT NOT NULL PRIMARY KEY,
    name VARCHAR(40) NOT NULL UNIQUE #Colour names should be unique
);

CREATE TABLE Player (
    pid INT NOT NULL PRIMARY KEY,
    name VARCHAR(40) NOT NULL,
    token INT NOT NULL UNIQUE,
    bank_balance INT NOT NULL DEFAULT 0,
    location INT NOT NULL DEFAULT 0,
    CONSTRAINT FK_token FOREIGN KEY (token)
        REFERENCES Token (tid),
    CONSTRAINT FK_location FOREIGN KEY (location)
        REFERENCES Location (lid)
);
```

```sql
CREATE TABLE Property (
    prid INT NOT NULL PRIMARY KEY,
    name VARCHAR(40) NOT NULL UNIQUE, #Unique property name
    cost INT NOT NULL DEFAULT 0 CHECK (cost >= 0), #Cost can't be negative
    colour INT NOT NULL,
    p_owner INT DEFAULT NULL,
    location INT NOT NULL UNIQUE, #UNIQUE to prevent any duplicate using the same location
    CONSTRAINT FK_colour FOREIGN KEY (colour)
        REFERENCES Colours (cid),
    CONSTRAINT FK_p_owner FOREIGN KEY (p_owner)
        REFERENCES Player (pid),
    CONSTRAINT FK_p_location FOREIGN KEY (location)
        REFERENCES Location (lid)
);

CREATE TABLE Turns (
    turn INT NOT NULL,
    #UNIQUE to prevent any player from being designated for more than 1 turn
    player_ID INT NOT NULL UNIQUE,
    #status = 0: playing, 1: IN JAIL, 2: defeated, 3: winner
    status INT NOT NULL DEFAULT 0,
    CONSTRAINT FK_t_player FOREIGN KEY (player_ID)
        REFERENCES Player (pid),
    PRIMARY KEY (turn , player_ID)
);

#Gamemanager table which gets the dice number and controls the game flow
#It is also to record the track of the gameplay as an audit table at the same time
CREATE TABLE Gamemanager (
    record_ID INT PRIMARY KEY AUTO_INCREMENT,
    round INT NOT NULL, #round adds up when every turn is played
    turn INT NOT NULL, #1 turn for 1 player
    dice INT NOT NULL, #dice number as an input
    location INT NOT NULL, #to track the location everytime a certain player moves and lands on it
    #to track the bank balance of a certain player after he plays his turn
    bank_balance INT NOT NULL,
    CONSTRAINT FK_turn FOREIGN KEY (turn)
        REFERENCES Turns (turn),
    CONSTRAINT FK_g_location FOREIGN KEY (location)
        REFERENCES Location (lid)
);

#gameView to show the status of the game at certain time
CREATE VIEW gameView AS
    SELECT (SELECT MAX(round) FROM Gamemanager) AS "Round", t.player_ID AS "Player No.",
        p.name AS "Player name", p.location AS "Location",
        group_concat(DISTINCT pr.name) AS "Properties owned", p.bank_balance AS "Bank balance",
        (CASE t.status WHEN 0 THEN "Playing" WHEN 1 THEN "IN JAIL" WHEN 2 THEN "Defeated"
        WHEN 3 THEN "Winner!" END) AS "Status"
    FROM Turns AS t
        INNER JOIN Player AS p ON t.player_ID = p.pid
        INNER JOIN GameManager AS gm ON gm.turn = t.turn
        LEFT JOIN Property AS pr ON t.player_ID = pr.p_owner
    GROUP BY t.turn;
```

```
DELIMITER $$

#Trigger to prevent more than 6 players from being added to the game
CREATE TRIGGER add_player
BEFORE INSERT ON Player FOR EACH ROW
BEGIN
    DECLARE cnt_player INT;
    SELECT
        COUNT(*)
    INTO cnt_player FROM Player;
    IF cnt_player >= 6 THEN
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'UP to 6 players can be registered per game.';
    END IF;
END$$

#Trigger to prevent bonus tiles from being located on a location for properties
CREATE TRIGGER b_location_chk_1
BEFORE INSERT ON Bonus FOR EACH ROW
BEGIN
        DECLARE type INT;
        SELECT Location.type INTO type FROM Location WHERE lid = NEW.location;
        IF type = 1 THEN
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Please check the location type';
        END IF;
    END$$

#Trigger to prevent bonus tiles from being located on a location for properties
    CREATE TRIGGER b_location_chk_2
    BEFORE UPDATE ON Bonus FOR EACH ROW
    BEGIN
        DECLARE type INT;
        SELECT Location.type INTO type FROM Location WHERE lid = NEW.location;
        IF type = 1 THEN
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Please check the location type';
        END IF;
    END$$

#Trigger to prevent property tiles from being located on a location for bonuses
    CREATE TRIGGER pr_location_chk_1
    BEFORE INSERT ON Property FOR EACH ROW
    BEGIN
        DECLARE type INT;
        SELECT Location.type INTO type FROM Location WHERE lid = NEW.location;
        IF type = 0 THEN
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Please check the location type';
        END IF;
    END$$

#Trigger to prevent property tiles from being located on a location for bonuses
    CREATE TRIGGER pr_location_chk_2
    BEFORE UPDATE ON Property FOR EACH ROW
    BEGIN
        DECLARE type INT;
```

```
        SELECT Location.type INTO type FROM Location WHERE lid = NEW.location;
        IF type = 0 THEN
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Please check the location type';
        END IF;
    END$$

DELIMITER ;

#Insert values to set the game
INSERT INTO Location  #(lid, type)
    VALUES (0, 0),
    (1, 1),
    (2, 0),
    (3, 1),
    (4, 0),
    (5, 1),
    (6, 0),
    (7, 1),
    (8, 0),
    (9, 1),
    (10, 0),
    (11, 1),
    (12, 0),
    (13, 1),
    (14, 0),
    (15, 1);

INSERT INTO Colours  #(cid, name)
    VALUES (0, "Yellow"),
    (1, "Green"),
    (2, "Orange"),
    (3, "Blue");

INSERT INTO Token  #(tid, name)
    VALUES (0, "Dog"),
    (1, "Car"),
    (2, "Battle-ship"),
    (3, "Top hat"),
    (4, "Thimble"),
    (5, "Boot");

INSERT INTO Player  #(pid, name, token, bank_balance, location)
    VALUES (0, "Jane", 1, 150, 13),
    (1, "Norman", 4, 250, 1),
    (2, "Mary", 2, 190, 8),
    (3, "Bill", 0, 500, 11);

INSERT INTO Property (prid, name, cost, colour, p_owner, location)
    VALUES     (0, "Kilburn", 120, 0, NULL, 1),
    (1, "Uni Place", 100, 0, 2, 3),
    (2, "Victoria", 75, 1, 3, 5),
    (3, "Piccadilly", 35, 1, NULL, 7),
    (4, "Oak House", 200, 2, 1, 9), #Original cost = 100
    (5, "Owens Park", 60, 2, 1, 11), #Original cost = 30
```

```
    (6, "AMBS", 400, 3, NULL, 13),
    (7, "Co-Op", 30, 3, 0, 15);

INSERT INTO Bonus  #(bid, name, description, location)
    VALUES (0, "GO", "Collect 200 pounds", 0),
    (1, "Chance 1", "Pay each of the other players 50 pounds", 2),
    (2, "IN JAIL", "must roll a 6 to get out", 4),
    (3, "Community Chest 1", "For winning a Beauty Contest, you win 100 pounds", 6),
    (4, "Free Parking", "No action", 8),
    (5, "Chance 2", "Move forward 3 spaces", 10),
    (6, "Go to Jail", "Go to Jail, do not pass GO, do not collect 200 pounds", 12),
    (7, "Community Chest 2", "Your library books are overdue. Play a fine of 30 pounds", 14);

INSERT INTO Turns  #(turn, player_ID, status)
    VALUES (0, 0, 0),
    (1, 1, 0),
    (2, 2, 0),
    (3, 3, 0);

#Session variable, total number of locations (tiles)
SET @no_of_tiles = (SELECT COUNT(*) FROM Location);
#Session variable, total number of active players of initial state
SET @init_no_of_players = (SELECT COUNT(*) FROM Player);
SET @turn = 0; #Session variable for turn number, turn starts from 0
#Session variable for the location of "IN JAIL" tile, now it is designated to location 4
SET @jail = (SELECT location FROM Bonus WHERE name = "IN JAIL");
SET @round = 1; #Session variable for round number, round starts from 1

DELIMITER $$

#dice trigger acts when gamemanager table gets a dice number inserted
CREATE TRIGGER dice_trigger
BEFORE INSERT ON Gamemanager FOR EACH ROW
#uses "before insert" trigger to insert the players landing location and bank balance together with the
#dice number for tracking
BEGIN
    DECLARE player_ID INT;
    DECLARE last_location INT;
    DECLARE last_bank_balance INT;

    SET NEW.turn = @turn, NEW.round = @round; #set the turn and round values to insert together
    #get the player_ID to move the player
    SELECT Turns.player_ID INTO player_ID FROM Turns WHERE Turns.turn = NEW.turn;

    #call the procedure move_player to move the player by the dice number
    CALL move_player(player_ID, NEW.dice);
    #gets the location the player finally landed on
    SELECT location INTO last_location FROM Player WHERE Player.pid = player_ID;
    #gets the players bank balance after he finishes his turn
    SELECT bank_balance INTO last_bank_balance FROM Player WHERE Player.pid = player_ID;
    #insert the location and bank balance together with the dice number INTO Gamemanager table
    SET NEW.location = last_location, NEW.bank_balance = last_bank_balance;

    #If the dice is not 6 then the next player plays
```

```
    IF (NEW.dice != 6) THEN
        CALL select_next_player(NEW.turn);
    END IF;

END$$

#procedure to move a certain player by a dice number
CREATE PROCEDURE move_player (player_ID INT, dice INT)
BEGIN
    DECLARE current_location INT;
    DECLARE status INT;

    SELECT location INTO current_location FROM Player
        WHERE Player.pid = player_ID;
    SELECT Turns.status INTO status FROM Turns
        WHERE Turns.turn = player_ID;

    #If the player in jail he has to roll a 6 to get out
    IF (status = 1 AND dice = 6)
    THEN
        UPDATE Turns SET status = 0 WHERE Turns.turn = player_ID;
    ELSEIF (status = 0) #If the player is not in jail then update his location
    THEN
        UPDATE Player
        #The player gets 200 pounds by going through "GO" tile
        SET bank_balance = IF(current_location + dice >= @no_of_tiles, bank_balance + 200, bank_balance)
            WHERE pid = player_ID;
        UPDATE Player
        #If else statement to adjust the location number when it goes over the total number of tiles
        SET location = IF(current_location + dice < @no_of_tiles, current_location + dice,
            current_location + dice - @no_of_tiles) WHERE Player.pid = player_ID;
        SELECT location INTO current_location FROM Player WHERE Player.pid = player_ID;
        #If the dice number is not 6 then the landing effect happens to the player
        IF (dice != 6) THEN CALL player_landing(player_ID, current_location);
        END IF;
    END IF;
END$$

#procedure to activates landing effects by checking the location type (0: bonus, 1:property)
CREATE PROCEDURE player_landing (player_ID INT, location INT)
BEGIN
    DECLARE location_type INT;

    SELECT type INTO location_type FROM Location WHERE lid = location;

    IF (location_type = 1) THEN
        #procedure to activate any effect by landing on a certain property
        CALL Landing_on_property(player_ID, location);
    ELSE
        #procedure to activate any bonus tile effect
        CALL Landing_on_bonus(player_ID, location);
    END IF;
END$$
```

```
#procedure to activate any property tile effect
CREATE PROCEDURE Landing_on_property (player_ID INT, location INT)
BEGIN
    DECLARE p_owner INT;
    DECLARE property_ID INT;
    DECLARE payable BOOL DEFAULT TRUE;
    DECLARE amount INT DEFAULT 0;
    #gets the property id the player landed on
    SELECT prid INTO property_ID FROM Property WHERE Property.location = location;
    #checks the owner of the property
    SELECT Property.p_owner INTO p_owner FROM Property WHERE Property.location = location;
    #gets the cost of the property
    SELECT cost INTO amount FROM Property WHERE Property.location = location;

    IF (p_owner IS NOT NULL) THEN #If the property is owned by someone,
        SELECT change_balance(p_owner, amount, FALSE) INTO payable; #the the player pays the rent
        #and the owner receives the rent
        SELECT change_balance(player_ID, -amount, FALSE) INTO payable;
    ELSE
        SELECT change_balance(player_ID, -amount, TRUE) INTO payable; #else,
        IF (payable) THEN #If he can actually afford to buy it,
            #Then he pays the cost and obtains the ownership
            UPDATE Property SET p_owner = player_ID WHERE Property.location = location;
            #monopoly_checker to check if he monopolises the colour
            CALL monopoly_checker(player_ID, location);
        END IF;
    END IF;
END$$

#Function to make changes equals to the "amount" to a certain players bank account and return bool value
#"payable" if he can actually pay the amount
CREATE FUNCTION change_balance (player_ID INT, amount INT, isBuyingProperty BOOL) RETURNS BOOL
DETERMINISTIC
BEGIN
    DECLARE balance INT;
    DECLARE isPayable BOOL DEFAULT TRUE;
    SELECT bank_balance INTO balance FROM Player WHERE pid = player_ID;
    #To check if the player can actually afford this change in his bank account
    SET isPayable = IF(balance + amount >= 0, TRUE, FALSE);

    IF (isPayable) #If he can afford it
    THEN #Then updates his bank account by the amount
        UPDATE player
        SET bank_balance = bank_balance + amount
        WHERE Player.pid = player_ID;
    ELSEIF (NOT isBuyingProperty) THEN #Else, if it is not to purchase a property,
        UPDATE player #Then the player goes bankrupt and inactive
        SET bank_balance = 0
        WHERE Player.pid = player_ID;
        UPDATE Property
        SET p_owner = NULL
        WHERE p_owner = player_ID;
        UPDATE Turns
        SET status = 2
```

```
        WHERE Turns.player_ID = player_ID;
        CALL winner_checker(); #winner_checker to see if there is only one player active
    ELSE BEGIN END; #If it is to buy a property, then nothing happens because he cant buy it
    END IF;
        #Return a bool value to use in landing_on_property procedure to change the ownership if TRUE
        RETURN isPayable;
END$$

#procedure to check if a certain player monopolises a colour, if so then it doubles the rent fees
CREATE PROCEDURE monopoly_checker(player_ID INT, location INT)
BEGIN
    DECLARE colour INT;
    DECLARE adj_p_owner INT;

    SELECT Property.colour INTO colour FROM Property WHERE Property.location = location;
    SELECT Property.p_owner INTO adj_p_owner FROM Property
        WHERE (Property.location != location AND Property.colour = colour);
    IF (player_ID = adj_p_owner) THEN
        #doubles the cost(rent fee) if it is monopolised
        UPDATE Property SET cost = cost * 2 WHERE Property.colour = colour;
    END IF;
END$$

#procedure to activate bonus effect
CREATE PROCEDURE landing_on_bonus (player_ID INT, location INT)
BEGIN
    DECLARE no_of_players INT;
    DECLARE bid INT;
    DECLARE payable BOOL;
    DECLARE amount INT DEFAULT 0;
    DECLARE amount2 INT DEFAULT 0;
    DECLARE idx INT;
    DECLARE temp_location INT;
    DECLARE dice INT;
    DECLARE status INT;
    SELECT Bonus.bid INTO bid FROM Bonus WHERE Bonus.location = location; #gets the bonus id
    #gets the number of the active players now
    SELECT COUNT(*) INTO no_of_players FROM Turns WHERE Turns.status != 2;

    CASE bid
        WHEN 1 THEN #bid = 1 for "Chance 1"
            SET idx = 0;
            SET amount = 50; #the amount the player has to pay to each of the other players active
            SET amount2 = amount * (no_of_players - 1);
            WHILE (idx < @init_no_of_players) DO
                SELECT Turns.status INTO status FROM Turns WHERE Turns.player_ID = idx;
                IF (idx = player_ID) THEN
                #the player who landed on the Chance 1 tile loses money
                SELECT change_balance(idx, -amount2, FALSE) INTO payable;
                ELSEIF (idx != player_ID AND status != 2) THEN
                #to give 50 pounds to each of the other players playing and undefeated
                SELECT change_balance(idx, amount, FALSE) INTO payable;
                END IF;
                SET idx = idx + 1;
```

```
            END WHILE;
        WHEN 5 THEN #bid = 5 for "Chance 2"
            SET dice = 3;
            CALL move_player(player_ID, dice); #Then moves the player 3 tiles ahead
        WHEN 3 THEN #bid = 3 for "Community Chest 1"
            #Then the player gets 100 pounds reward
            SELECT change_balance(player_ID, 100, FALSE) INTO payable;
        WHEN 7 THEN #bid = 7 for "Community Chest 2"
            SELECT change_balance(player_ID, -30, FALSE) INTO payable; #Then the player pays 30 pounds
        WHEN 6 THEN #bid = 6 for "Go to Jail"
            #Then the player is sent to jail not receiving the "Go" 200 pounds bonus
            UPDATE Player SET location = @jail WHERE Player.pid = player_ID;
            UPDATE Turns SET status = 1 WHERE Turns.turn = player_ID;
        WHEN 2 THEN #bid = 2 for "IN JAIL"
            #Then the player gets detained in jail
            UPDATE Turns SET status = 1 WHERE Turns.turn = player_ID;
            ELSE BEGIN END;
    END CASE;
END$$

#procedure to set the turn and round numbers
CREATE PROCEDURE select_next_player (current_turn INT)
BEGIN
    DECLARE status INT DEFAULT 2;
    DECLARE temp_turn INT DEFAULT 0;
    SET temp_turn = current_turn;
    WHILE (status = 2) DO #To skip any defeated player's turn
        SET temp_turn = temp_turn + 1;
        IF (temp_turn = @init_no_of_players) THEN
            SET @round = @round + 1,
            temp_turn = 0;
        END IF;
        SELECT Turns.status INTO status FROM Turns WHERE Turns.turn = temp_turn;
    END WHILE;
    SET @turn = temp_turn;
END$$

#procedure to check if someone wins the game
CREATE PROCEDURE winner_checker ()
BEGIN
    DECLARE no_of_players INT;
    DECLARE winner_ID INT;
    SELECT (COUNT(turn)) INTO no_of_players FROM Turns WHERE Turns.status != 2;
    IF (no_of_players = 1) THEN
        SELECT turn INTO winner_ID FROM Turns WHERE Turns.status != 2;
        UPDATE Turns SET status = 3 WHERE Turns.turn = winner_ID;
        UPDATE Player SET name = CONCAT(name, " (Winner)") WHERE pid =
            (SELECT player_ID FROM Turns WHERE Turns.turn = winner_ID);
    END IF;
END$$

DELIMITER ;
```

4.3. Gameplay and gameView

Figure 5 below is to make the “gameView” to show each player’s ID, name, location, bank balance, properties owned, and status with the round number.

Figure 6. Queries to make the “gameView”

```
CREATE VIEW gameView AS
    SELECT (SELECT MAX(round) FROM Gamemanager) AS "Round", t.player_ID AS "Player No.",
        p.name AS "Player name", p.location AS "Location",
        group_concat(DISTINCT pr.name) AS "Properties owned",
        p.bank_balance AS "Bank balance",
        (CASE t.status WHEN 0 THEN "Playing" WHEN 1 THEN "IN JAIL" WHEN 2 THEN "Defeated" WHEN 3
        THEN "Winner!" END) AS "Status"
    FROM Turns AS t
        INNER JOIN Player AS p ON t.player_ID = p.pid
        INNER JOIN GameManager AS gm ON gm.turn = t.turn
        LEFT JOIN Property AS pr ON t.player_ID = pr.p_owner
    GROUP BY t.turn;
```

The queries and gameView for the gameplay round 1 are as below.

Figure 7. Round 1 queries

```
INSERT INTO Gamemanager (dice)
    VALUES (3); #Jane rolls a 3, gets 200 pounds bonus for landing on "GO"

INSERT INTO Gamemanager (dice)
    VALUES (1); #Norman rolls a 1, spends 150 pounds to pay 50 pounds each to the other 3 players

INSERT INTO Gamemanager (dice)
    VALUES (4); #Mary rolls a 4, sent to jail

INSERT INTO Gamemanager (dice)
    VALUES (2); #Bill rolls a 2, buys "AMBS" by paying 400 pounds

SELECT * FROM gameView;
```

Figure 8. Round 1 gameView

Round	Player No.	Player name	Location	Properties owned	Bank balance	Status
1	0	Jane	0	Co-Op	400	Playing
1	1	Norman	2	Oak House,Owens Park	100	Playing
1	2	Mary	4	Uni Place	240	IN JAIL
1	3	Bill	13	AMBS,Victoria	150	Playing

The queries and gameView for the round 2 are as below.

Figure 9. Round 2 queries

```
INSERT INTO Gamemanager (dice)
    VALUES (5); #Jane rolls a 5, pays Bill 75 pounds for landing on "Victoria"

INSERT INTO Gamemanager (dice)
    VALUES (4); #Norman rolls a 4, gets 100 pounds by landing on "Community Chest 1"

INSERT INTO Gamemanager (dice)
    VALUES (6); #Mary rolls a 6 and gets out from jail

INSERT INTO Gamemanager (dice)
    VALUES (5); #Mary rolls a 5, pays Norman 200 pounds for landing on "Oak House"

INSERT INTO Gamemanager (dice)
    VALUES (6); #Bill rolls a 6 and gets no landing effect by R5, gets 200 pounds "GO" bonus

INSERT INTO Gamemanager (dice)
    VALUES (3); #Bill gets his turn again and rolls a 3, gets 100 pounds by "Community Chest 1"

SELECT * FROM gameView;
```

Figure 10. Round 2 gameView

Round	Player No.	Player name	Location	Properties owned	Bank balance	Status
2	0	Jane	5	Co-Op	325	Playing
2	1	Norman	6	Oak House,Owens Park	400	Playing
2	2	Mary	9	Uni Place	40	Playing
2	3	Bill	6	AMBS,Victoria	525	Playing

REFERENCES

- GeeksforGeeks (2022). *CONCAT() function in MySQL*. Available at: https://www.geeksforgeeks.org/concat-function-in-mysql/ (Accessed: 8 November 2022).
- MySQLTUTORIAL (2022). *MySQL CASE Statement*. Available at: https://www.mysqltutorial.org/mysql-case-statement/ (Accessed: 8 November 2022).
- MySQLTUTORIAL (2022). *MySQL Stored Procedure Variables*. Available at: https://www.mysqltutorial.org/variables-in-stored-procedures.aspx (Accessed: 8 November 2022).
- Oracle (2022). *MySQL 8.0 Reference Manual: 13.1.17 CREATE PROCEDURE and CREATE FUNCTION Statements*. Available at: https://dev.mysql.com/doc/refman/8.0/en/create-procedure.html (Accessed: 8 November 2022).
- Oracle (2022). *MySQL 8.0 Reference Manual: 13.7.6.1 SET Syntax for Variable Assignment.* Available at: https://dev.mysql.com/doc/refman/8.0/en/set-variable.html (Accessed: 8 November 2022).
- Oracle (2022). *MySQL 8.0 Reference Manual: 25.3.1 Trigger Syntax and Examples.* Available at: https://dev.mysql.com/doc/refman/8.0/en/trigger-syntax.html (Accessed: 8 November 2022).
- W3resource (2022). *MySQL CONSTRAINT.* Available at: https://www.w3resource.com/mysql/creating-table-advance/constraint.php (Accessed: 8 November 2022).
- W3resource (2022). *MySQL IF() function.* Available at: https://www.w3resource.com/mysql/control-flow-functions/if-function.php (Accessed: 8 November 2022).