Design document

MDW – Ludo game

FONTYS UNIVERISTY OF APPLIED SCIEENCES

May 15, 2016

Authored by: Monica Stoica, Dimitar Vikentiev, Rosen Danev

Table of Contents

Architecture diagram 2

Description of Interfaces 3

IGame: 3

IGameCallback: 3

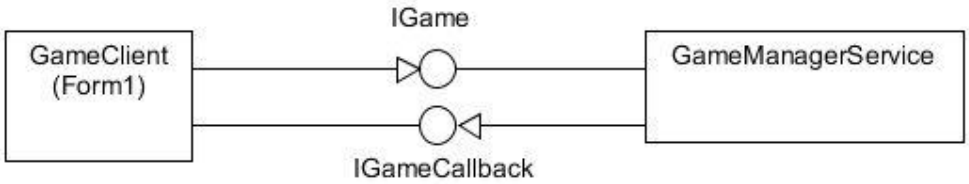
Class Diagram 4

Sequence diagrams 6

- Start the game 6
- Throw a die 7
- Move pawn 8
- Pause the game 9
- Quit the game 9

Database diagram 10

Architecture diagram



Description of Interfaces

IGame:

void Connect():

This method will be called each time when a person enters the game and that user will be connected. His/her information, such as username, will be shown in a certain list box in the main form.

void Disconnect():

This method one will be used when a player exists the game and the he/she will be disconnected. The player will also be removed from the list box and the other players will be informed.

string GetPlayerInfo(string username):

It will return a string with all information of a certain player depending on his/her username.

int ThrowDice():

Will return a random integer from 1 to 6.

void MoveForward(int diceresult):

When this method is called the pawn will move forward according to the diceresult.

IGameCallback:

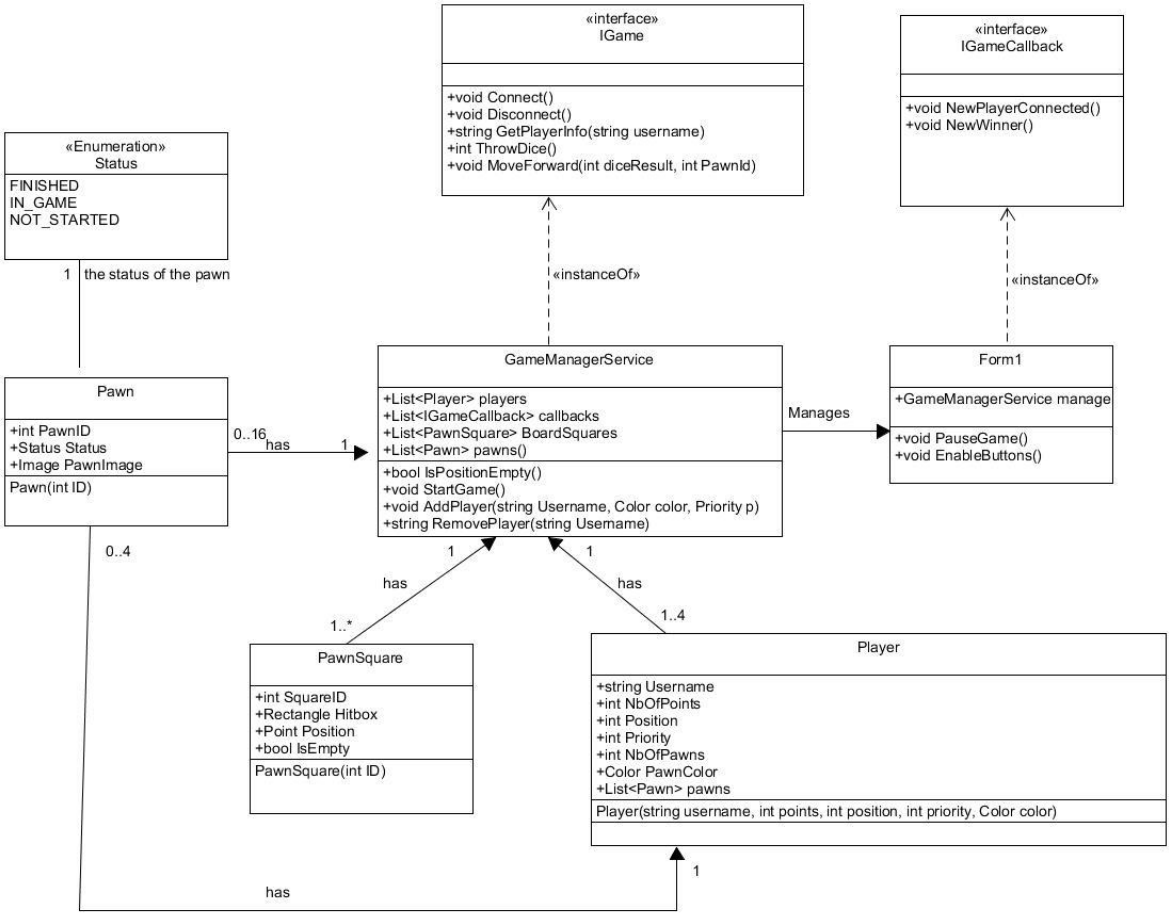
void NewPlayerConnected():

This method will inform other players when a new player is connected.

void NewWinner():

The NewWinner method will inform other players when some of them has finished and will show his/her place (ranking) among other players.

Class Diagram



Below, you can find a description of the fields, properties and methods used in our Class diagram.

Enumeraton Status{ FINISHED, IN_GAME, NOT_STARTED}

Class Player

Properties:

- Username: string – represents the name of the player
- NbOfPoint: int – represents the number of points accumulated by a player
- Position: int – represents the position of the player compared to other players
- Priority: int – represents the priority of the player when starting a new game (e.g. who will roll the die first)
- NbOfPawns: int – represents the number of pawns the player needs to bring to the finish line in order to win
- PawnColor: int – represents the colour of the pawns of a player
- Pawns : List<Pawn> - stores all the pawns of a player

Constructor:

- Player(string username, int points, int position, int priority, Color color)

Class Pawn

Properties:

- PawnID: int – the unique identification of a pawn
- Status: status – the current status of a pawn
- PawnImage: Image – the representative image of a pawn

Constructor:

- Pawn(int ID);

Class PawnSquare

Properties:

- SquareID : int – unique identification of each square
- Hitbox: rectangle – represents the exact square
- Position: Point – the X and Y coordinates of the top left corner of the rectangle
- isEmpty: bool – 'True' if there is no pawn on that square. Otherwise, false.

Constructor:

- PawnSquare(int ID)

Class GameManagerService

Properties:

- Players: List<Player> - list of all players
- Callbacks: List<IGameCallBack> - list with all callbacks
- BoardSquares: List<PawnSquare> - list with all squares
- Pawns: List<Pawn> - list with all pawns

Methods:

- IsPositionEmpty: bool - 'True' if there is no pawn on that square. Otherwise, false.
- StartGame: void – checks how many players are in the game. If there are more than 2 and less than 4, then the game will start.
- AddPlayer: void – when a new player is connected, he/she will be added to the list
- RemovePlayer: string – when a user quits the application, he/she will be removed from the list

Class Form1

Properties:

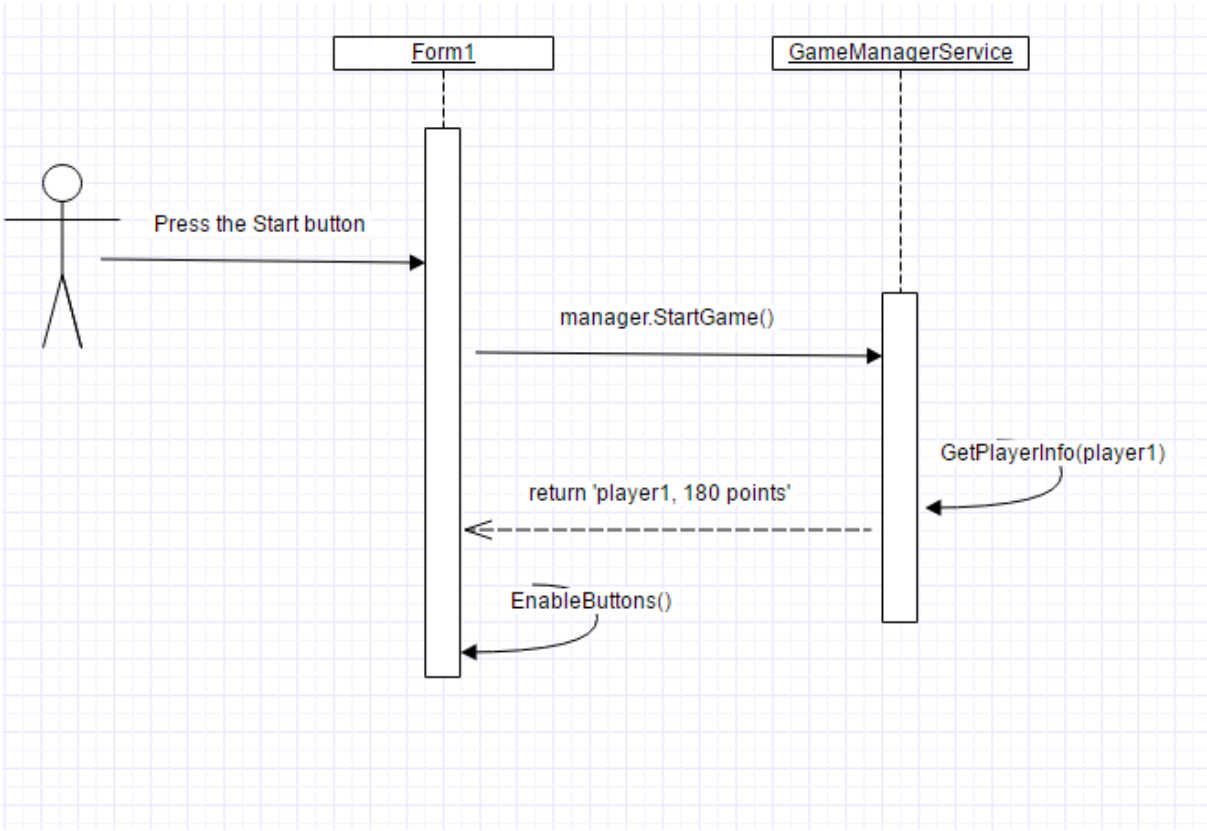
- Manager: GameManagerService - instance from the service

Methods:

- PauseGame: void – will pause the game by disabling all the buttons
- EnableButtons: void – enables the buttons so that the game can start

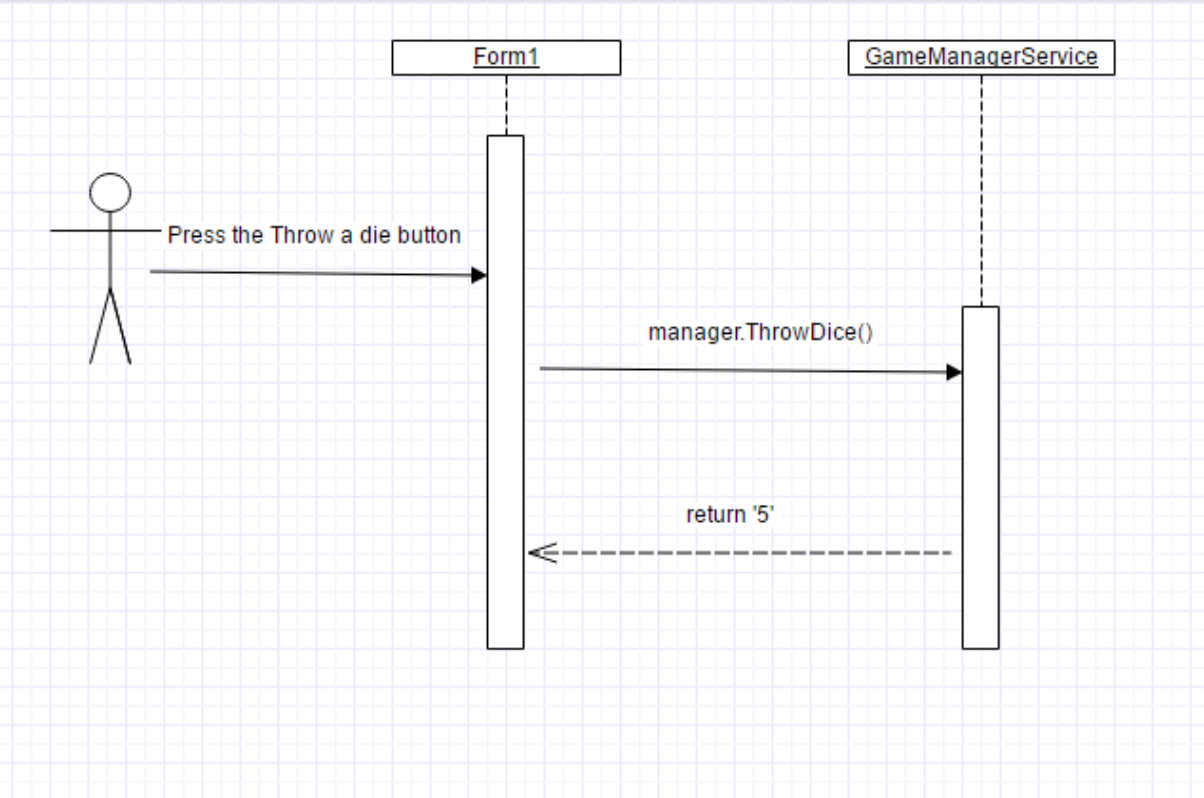
Sequence diagrams

Start the game



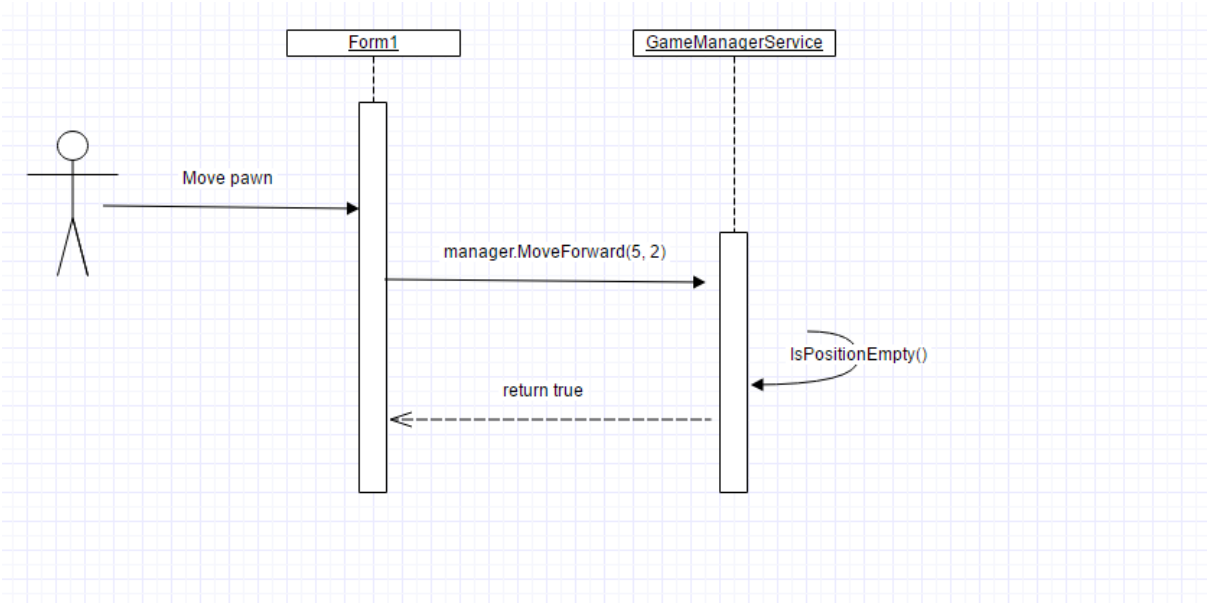
In order to start the game, the user with the highest priority (therefore the user who entered first the game) has the right to start the game. The system checks if there are at least two players, displays the information about each player and enables the 'Throw a die', 'Quit' and 'Pause' buttons.

Throw a die



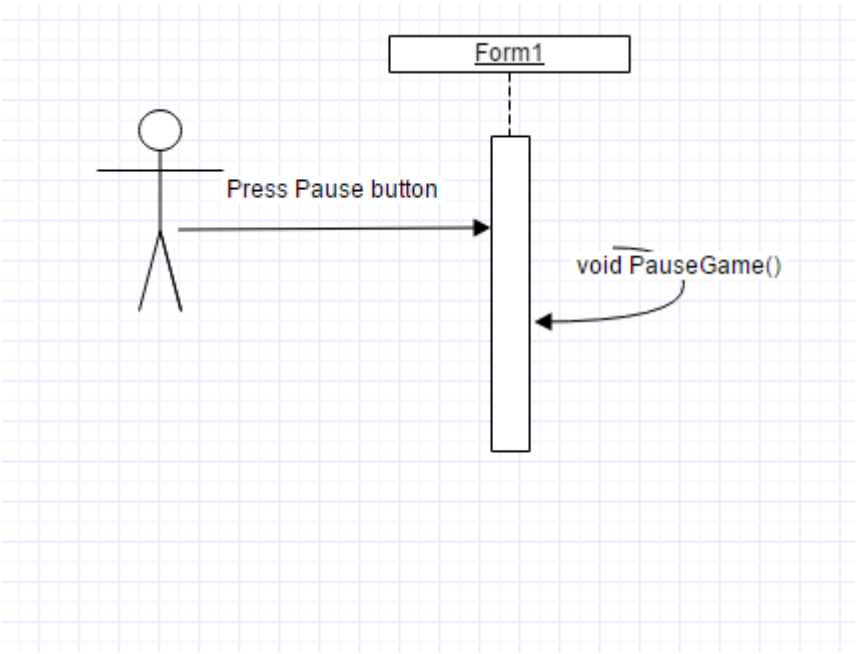
When his/her turn comes, a user has to throw a die. To do so, he/she presses the 'Throw a die' button. The system generates a random number between 1 and 6 and informs the player about the result.

Move pawn



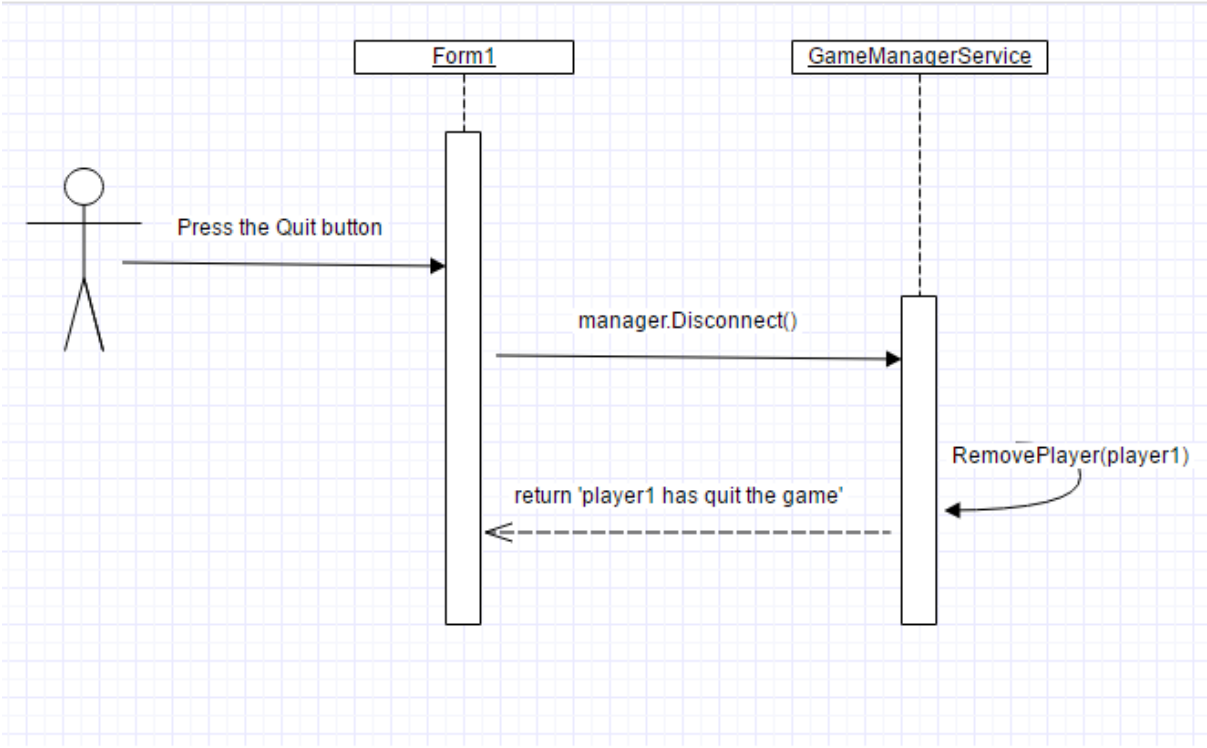
After the user has thrown a die, it is time to move the pawn. The system checks if the position on which the pawn is supposed to be next (5 represents the number drawn and 2 represents the id of the pawn) is free. If yes, the system returns true and indicates the user where to move the pawn. If the position is not free then the initial pawn will be spawned and replaced by the new pawn.

Pause the game



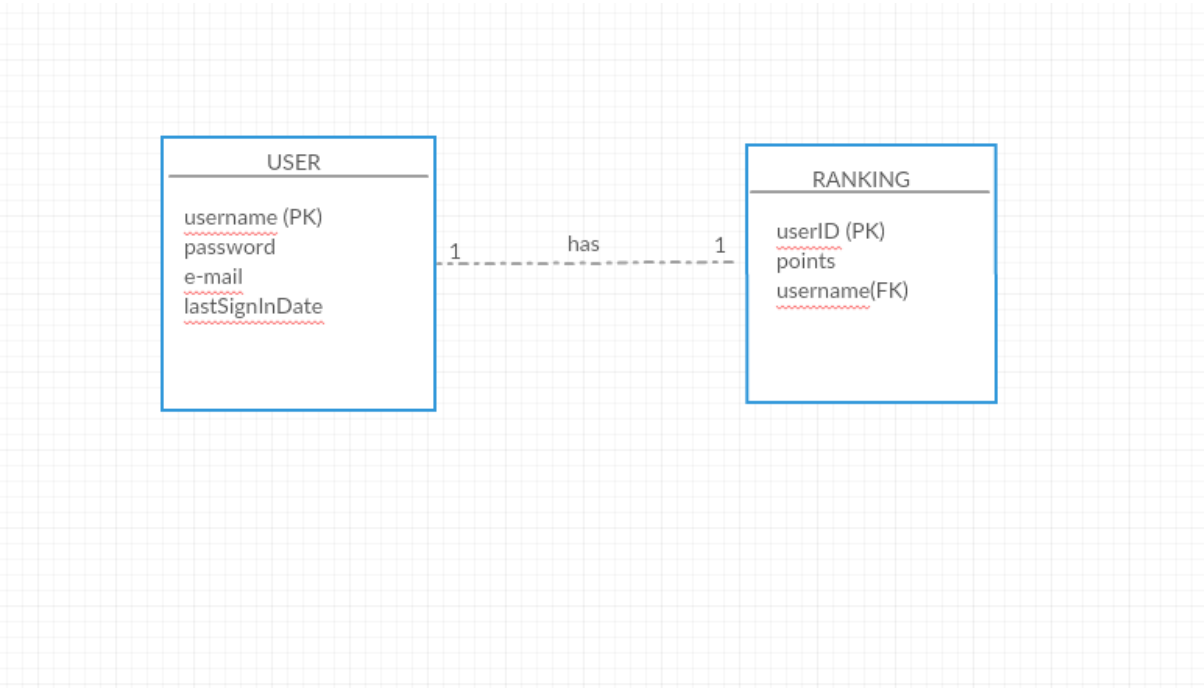
When the game is running, a player has to opportunity to pause the game. The PauseGame() method will only disable the buttons. Therefore, the player can only quit the game but he cannot continue to play until the game is resumed by the player who has paused.

Quit the game



At any point, the user can choose to exit the game by pressing the 'Quit' button. The system will disconnect him/her and the other players will be informed that the user has left the game

Database diagram



The diagram above represents the database design which will be used in order to implement the Sign in, Log in and Ranking functionalities. The entity on the left, USER, stores the information about a user such as username(string), password (string), e-mail(string), lastSigInDate (DateTime). The primary key is username since there cannot be two players with the same username.

The RANKING entity will store information about a player's performance such as points(int), username(FK) as a foreign key and userID(int) as an auto-incremented primary key.

The RANKING entity depends on the USER entity and the relationship between the two entities is one-to-one. (One user can have only one ranking and a ranking belongs to exactly one user).