Topic and description of the project

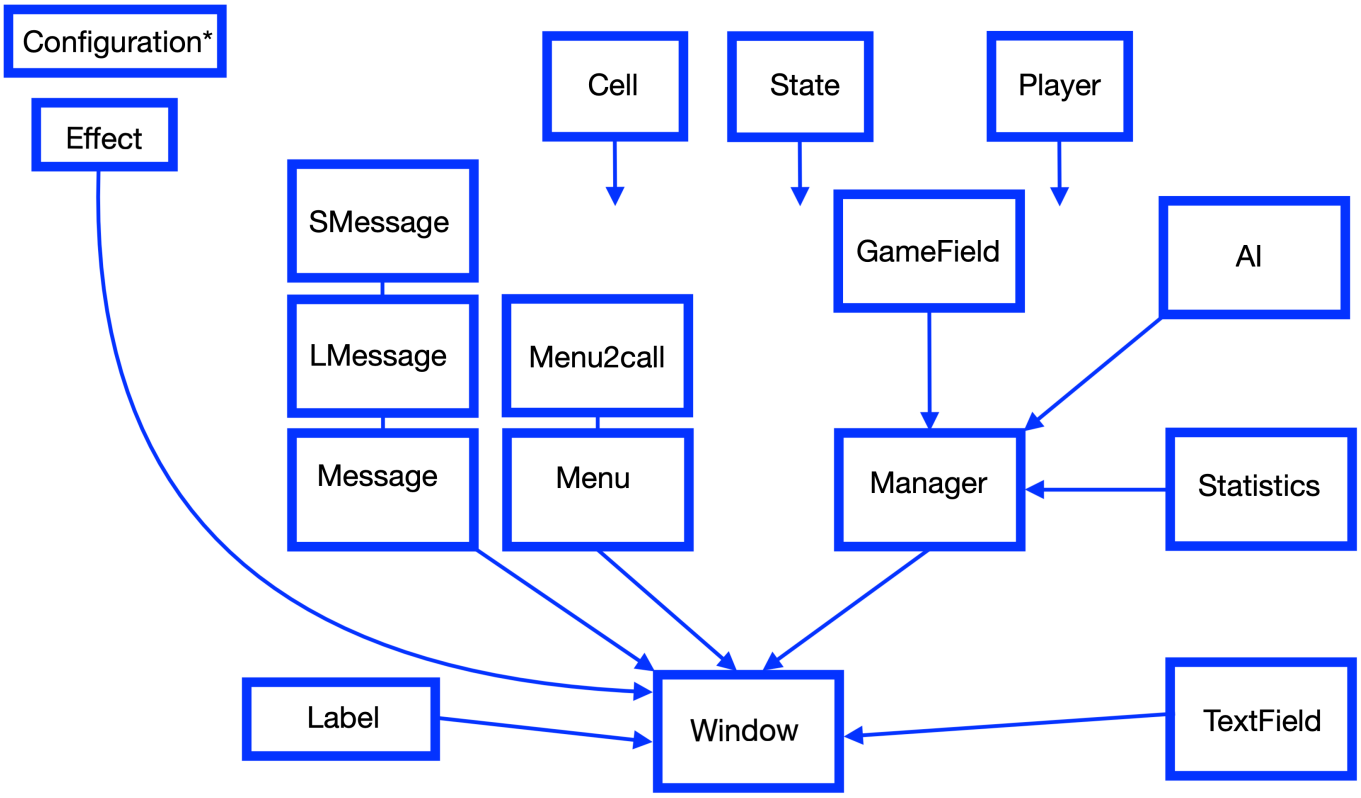
This is a popular tic-tac-toe game that was created using the Pygame library. This game has an intuitive graphical interface. You can play in classic mode 3x3 with your friend or with AI. You can play on a big field (from 4x4 to 10x10) with your friend (the possibility of playing with AI on a big-sized field is planned in the future). You can write your name in the game and your friend's name in the special text field (to do it you need just type on your keyboard and push a button, a text save automatically). However, if you don't write anything the program'll write down a name automatically. During the game nice music is playing but if you want to turn off it you'll push the "sound off" button (the "sound on" button is next to it). Your progress, AI's progress, and your friend's progress are saved in a general CSV file for all sizes as in a "special size" file. If a CSV file doesn't exist the program creates it by itself. The Window.py is the main file. Pleaser gaming!

The use of possible external libraries

This game uses the Pygame library, the Sys (syspath) and the Threading (thread6) modules.

The solution principle of the project

This project consists of GUI and back-end parts. The code of the game was written using the OOP style and includes 15 classes which are divided into different files (one file for each class).



Scheme of class structures

Descriptions of the classes:

1. WINDOW.

The Window is the main class. He starts the game process and game objects such as buttons, labels, messages, text fields, the game field (GUI objects), and the game manager (Manager).

2. MANAGER

This class rules the game process and initializes the GameField class (GUI object).

3. GAMEFIELD

To create a game board where you and your friend can put symbols.

4. TEXTFIELD

This class provides a possibility to write down your and your friend's names. He creates a GUI object - a text field.

5. NUMBERTEXT

This class is a child of TextField but allows a player to write only numbers from 4 to 10 (if a player writes a wrong number that standard game 3x3 is created).

6. STATISTICS

The Statistics class realizes writing and reading statistics data from and into CSV files and also saves local players' data such as count and quantity of victories, draws, and defeats.

7. AI

The AI class is a class to play games with AI, which is based on the MiniMax algorithm with a-b pruning.

8. LABEL

This class creates a GUI object with a label in a menu.

9. MESSAGE

This class creates a GUI object with a count which displays during a game.

10.LMESSAGE

This class is a child of Label but he creates special labels on a new transparent lay with centering.

11. SMESSAGE

This class is a child of Label but he creates special labels on a new color lay in a menu.

12. MENU

The Menu class creates buttons to call one function by clicking.

13. MENU2CALL

This class is a child of Menu but allows to call two functions at the same time.

14. CELL

This class is class about a cell's state: empty - "Void", cross - "Cross", and zero - "Zero".

15. STATE

The state class saves states about a local game-winner (Win_P1 and Win_P2) or draw (DRAW).

16. PLAYER

This class saves information about current gamers.

17. CONFIGURATION

This file isn't a class, it consists of configuration information such as color, sizes, images, etc.

18. EFFECT

This file has the "fade_to_black" function which fades a screen during transition between menu sections.

As you can see in the scheme above, Cell, Player, and State classes are used in some different classes simultaneously.
To solve the task of writing work code SOLID principles were.
In the code, the field is presented as a matrix (two-dimensional array). The game creates that GIU game object that is drawn according to the state of the matrix.

Functions and their relationships

In the OOP paradigm commonly used "method" is used instead of "function" but because the instruction contains the name "function" I'm gonna use the "function" word meaning “method”.

Functions of the Window class

1. First of all, the Window class initializes. Initializing the class calls initializing the Pygame, music, font, variables, lists, and the first page of the menu.
2. Then the “run” function is called. This function generates of game process in an endless cycle. Run calls also event, draw, hover_check, and main_loop functions. Updating of the screen happens every second according to the Pygame clock function and variable as FPS = 60. Function such as event, draw, hover_check, and main_loop calls all the time but trigger only that way if the corresponding list is filled.
3. Event is a function that checks events like mouse button down and keys down.
4. Draw function is drawing GUI objects and fade effect.
5. Hover_check checks the hovered state of buttons.
6. Main_loop is the main function of the process related to the game. if the game stops this function calls a Thread (to create one more parallel program flow) with a delay of 2.5 seconds (it needs to create a smooth transition between the game ending and the menu) if the game finishes.
7. The cleaner function cleans all the lists. When a list is empty then a GUI object deletes.
8. Create_menu_start_1 and similar to it other functions (except create_field) need to create the menu pages. They call each other by a button click. They also added variables of GUI objects into lists for displaying.
9. Create_field function generates the Manager class object and the Message class object. If a user wants to play on a big-size field this function creates the Manager object expressly indicating an n-variable (size of the field) or if it’s a standard size writes None instead of it.
10. Top_list creates a page with statistics about the top 10 players of the game.

Functions of the Manager class

1. If an n-variable exists then width, height, and margin are gonna be calculated according to the n-size in the opposite case it’s gonna be the standard value according to class variables. Also, the surface for drawing is created here. Then the matrix denoting the field is also created here. The last point is creating the GUI GameField object with all the geometry parameters of the cell.
2. Main_loop is the main function that rules the game. It checks who is a winner or the game finished as a draw, calls the hover_check, the draw, and the update function in the GameField class (for GUI objects drawing), and calls the ai_mode during AI mode.
3. Hover_check checks a hover state for a cell. If the coordinates are in the right diapason the analogous function in the GameField calls in that case.
4. Ai_mode calls if the game mode needs an AI algorithm. If the first player is AI and it’s his move (or the second player is AI and it’s his move) the AI function is called. When

this function returns her move (coordinates of a move) ai_mode calls ai_round to change a cell of the matrix.
5. Handler is a function that is called from the Window class event if a gamer clicks on any cell. This function changes this cell in the matrix according to the coordinates.
6. Count is a static method that generates a text about the ending of the game and sends it back.
7. Checker checks the state of the game: the winner is one of the gamers or a draw.

Functions of the GameField class

1. The initialized function creates the class's properties (width, height, margin of the cell and n-size of the matrix, surface for drawing).
2. Draw function draws the field.
3. Update function changes the field if any cell has changed.
4. Hover_check highlights a cell where a mouse pointer is.
5. Win_way draws the winner cells combination in the field.
6. Some auxiliary functions are.

Functions of the AI class

1. Main cycle puts the AI symbol in all the void cells and calls MiniMax for each. If a new count is better this way is chosen.
2. MiniMax function based on MiniMax algorithm with a-b pruning. It's a recursive function that counts all possible states of the game and returns all ways. It uses Minimax_checker.
3. Minimax_checker is a function. It returns 10 if AI wins, -10 in the opposite case, and 0 if it's a draw.

Functions of the Statistics class

1. Counter is a function to count and save points.
2. Reset function is used to update the count for a new player or new players.
3. Reader is a read information function that opens a needed file.
4. Writer and Writer_all functions need to write the info about players after every game. (Writer for special field size and Writer_all for general table). Both these functions check the files (does it exist? If no that creates a new one) and check the content (does it exist? If no write new information). If the files exist and they aren't empty the info is rewritten and updated.
5. Some auxiliary functions are.

Functions of the Message/LMessage/SMessage classes

All these classes are similar and have similar functions such as init and draw. But Message creates a simple text object on a color background, LMessage creates a special label on a new transparent lay and aligns labels at the center of the window, and SMessage on a white lay.

Functions of the Label class

This class has two functions: init and draw. Draw creates simple labels for menu pages.

Functions of the TextField/NumberText class

They are similar classes with similar methods however NumberText allows a player to write only numbers (allows a player to write only numbers from 4 to 10) and checks that this number (if a player writes a wrong number that standard game 3x3 is created).

1. Enter function writes a new symbol or deletes the last.

2. Draw shows a text field (rectangle with letters).
3. Write_size check a size number.

Functions of the Menu and Menu2call classes

1. Draw function shows a button
2. Hover_check checks the hover state and changes the background picture.
3. Handle_click calls a function that connects to the button (lambda function).

Menu2call has only one difference, this type of button has 2 lambda functions.

Functions of the Effect file

This file contains the fade function which draws the black background lay that becomes darker and darker.

Special variables

Some classes have special variables.

1. click (GameField)

This variable needs to player's click don't draw while the game isn't created.

2. stop (GameField)

This variable signals that Main_loop must be finished and must return the text to the Window class.

3. manager (Window)

This variable contains the Manager class object.

4. ending (Window)

If this variable is True the game is finished and the menu is drawn.

5. fade (Window)

If this variable is True that fade_to_black draws.

How the game could be improved?

1. First of all, creating a fast algorithm (for instance MTD(f)) to play with AI on a big field (from 4x4 to 10x10).
2. The second thing is a saving data system. Now, it's a simple CSV file (according to the requirements of the project), however, if the game scales it'll need to upgrade this system (create SQL DB and special tools for effective communication with this DB). If the game becomes a web game it'll require API and his setting.