Technical Documentation

1. Introduction

This is the technical documentation of our version of the Yinsh game. Coded with Python and displayed with the library Tkinter.

2. Required Libraries

Python installation : https://www.python.org/

Library installation : Go to your cmd and enter the following command : pip install <LibraryName>

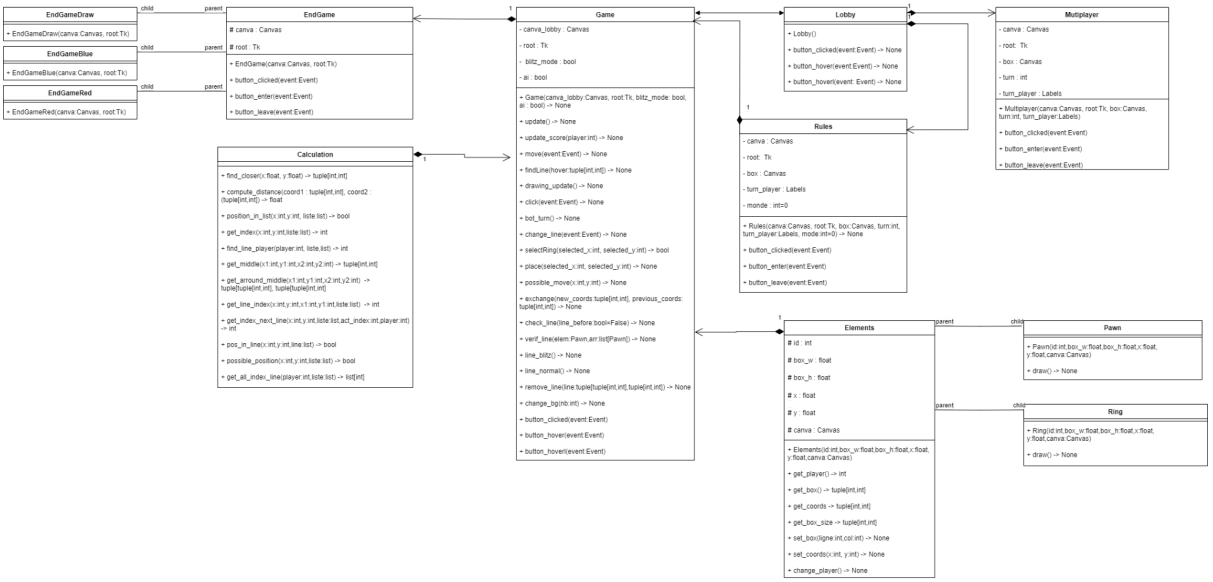
Install the following libraries : tkinter, math, pillow and pygame. You may already have at least one of these libraries

3. Reasons for the choice of technology

For this project, Python and C seemed to be the most recommended languages to use. In this case, we chose Python because we know the language well and it seems to be more convenient because we were using some classes in the way we thought of the project. For the visual interface, we chose the library Tkinter because it is the most intuitive visual interface library.

4. Description of the data structures

Here is a UML diagram representing the data structures. You can download it by clicking on the link linked to the image.

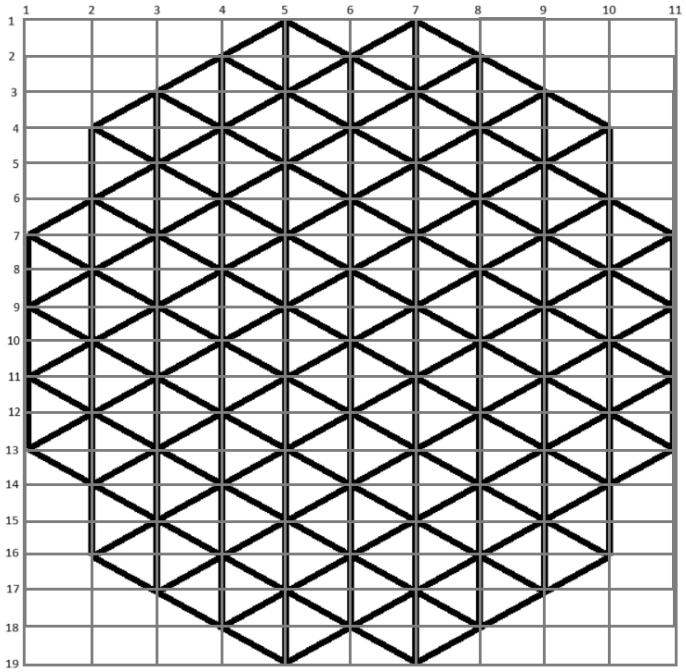


Here is a schema of how the project files are organized.

```
projet_yinsh/
│
├── Documentation/
│      ├── User_Documentation.pdf
│      └── Technical_Documentation.pdf
└── project/
      ├── img/
      │      ├── bg/
      │      ├── buttons/
      │      └── circles/
      ├── music/
      │      └── igmusic.mp3
      ├── Calculation.py
      ├── Elements.txt
      ├── EndGame.py
      ├── Game.py
      ├── Lobby.py
      ├── Multiplayer.py
      └── Rules.py
```

5. Algorithms for the handling of the grid

To handle the grid we decided to not have a grid strictly speaking we do all the placement by using coordinates depending on the size of the user screen in order to be able to use the game on any screen. All the elements have their own coords and are displayed using those. We can see on that picture just below the coords depending on the size of the grid



6. Algorithms for the movements of the rings

To collect the input of the player we use the tkinter bind method with the left click and with that we use the find_closer function to have the coords of the closest point around.
To move the rings we had to modify their coordinates using the set_coords or set_box method to modify the place of the rings and then by drawing them again they'll not be at the same place. At the same time the exchange method will change the owner of each pawn between the previous place and the new place using the change_player method.

7. Algorithms for the detection of lines

Everytime a ring moves a method called check_line is called and this method will call the verif_line method for every pawn. the verif_line method verify for a pawn if there is four pawn below him or on a diagonal on its right, if there is, a line if added in a attribute of the game that is a list and the check_list method calls either the line_blitz if the gamemode is blitz or the line_normal else.

8. Algorithms for the choice of the lines

For that part we used the hover with the informations given by the tkinter bind method put on Motion we can have the coordinates of the mouse we then compute everytime the coords of the closest point with the find_closer function and if the closer is different than the previous closer then we would change the previous one to the new one and then we call find_line to find a line with that point.
In that method there is some kind of ranking of the lines, if the point hovered is the center of a line then the line will be the most important after that case the second important is the line with the point as one of the inside point, and then it's the category where the point is on an extremity of the line.
In our game we also have another input for the user, if the user right-clicks when they are on a point will change the line displayed for another line crossing that point, for that we collect the index of the line displayed and searching in the list for line of the same player with an other index and the line must be crossing the point.

9. Process used to link and communicate between the two computers in online mode

To link the computers in online mode, we need to use the socket and threading libraries. Socket is used to connect the computers and threading is needed to start multiple loops at a time. The socket needs to gather 2 pieces of information, the ip adresse and the port. If 2 computers on the same network put the same ip adresse (the server's one) and the same port, they can connect together. Now for the game, we can send the information from one computer to another to make the game playable online by exchanging each turn the information needed.

10. Function explanation

10.1. Lobby.py

Class Lobby
It is the lobby where the player can choose the game mode and start the game.
In this class, there are mainly Tkinter methods used.

Method __init__(self)->None
Constructor of the class Lobby
There are defined, a big part of the variables.
Set up the Tkinter window settings.

Method blitz_button_clicked(self,event :Event) ->None
Use the Game Class to start a game in blitz mode

Method bot_button_clicked(self,event :Event) ->None
Use the Game Class to start a normal game against an AI

Method leave_button_clicked(self,event :Event) ->None
Close the game

Method normal_button_clicked(self,event :Event) ->None
Use the Game Class to start a normal game

Method players_button_clicked(self,event :Event) ->None
Use the Game Class to start a normal game for 2 players son the same monitor

Method return_1_button_clicked(self,event :Event) ->None
Delete the 2 players, bot and return button to display the normal, blitz and return buttons instead.

Method return_2_button_clicked(self,event :Event) ->None
Delete the normal, blitz and return button to display the play,rules and leave buttons instead.

Method rules_button_clicked(self,event :Event) ->None
Use the Rules Class to open a window with the rules of the game

Method start_button_clicked(self,event :Event) ->None
Delete the play, rules and leave button to display the 2 players, bot and return buttons instead.

Method normal_button_hover(self,event :Event) ->None
Add a hover effect on the normal button.

Method normal_button_hoverl(self,event :Event) ->None
Disable the hover effect on the normal button.

Method blitz_button_hover(self,event :Event) ->None
Add a hover effect on the blitz button.

Method blitz_button_hoverl(self,event :Event) ->None
Disable the hover effect on the blitz button.

Method rules_button_hover(self,event :Event) ->None
Add a hover effect on the rules button.

Method rules_button_hoverl(self,event :Event) ->None
Disable the hover effect on the rules button.

Method players_button_hover(self,event :Event) ->None
Add a hover effect on the 2 players button.

Method players_button_hoverl(self,event :Event) ->None
Disable the hover effect on the 2 players button.

Method bot_button_hover(self,event :Event) ->None
Add a hover effect on the bot button.

Method bot_button_hoverl(self,event :Event) ->None
Disable the hover effect on the bot button.

Method start_button_hover(self,event :Event) ->None
Add a hover effect on the play button.

Method start_button_hoverl(self,event :Event) ->None
Disable the hover effect on the play button.

Method leave_button_hover(self,event :Event) ->None
Add a hover effect on the leave button.

Method leave_button_hoverl(self,event :Event) ->None
Disable the hover effect on the leave button.

Method return_1_button_hover(self,event :Event) ->None
Add a hover effect on the return button.

Method return_1_button_hoverl(self,event :Event) ->None
Disable the hover effect on the return button.

Method return_2_button_hover(self,event :Event) ->None
Add a hover effect on the return button.

Method return_2_button_hoverl(self,event :Event) ->None
Disable the hover effect on the return button.

Method return_3_button_clicked(self,event :Event) ->None
Delete some buttons and add some other

Method return_3_button_hover(self,event :Event) ->None
Add a hover effect on the return button.

Method return_3_button_hoverl(self,event :Event) ->None
Disable the hover effect on the return button.

Method return_4_button_hover(self,event :Event) ->None
Add a hover effect on the return button.

Method return_4_button_hoverl(self,event :Event) ->None
Disable the hover effect on the return button.

Method local_button_clicked(self,event :Event) ->None
Disable the hover effect on the return button.

Method online_button_clicked(self,event :Event) ->None
Disable the hover effect on the return button.

Method local_button_hover(self,event :Event) ->None
Add a hover effect on the return button.

Method local_button_hoverl(self,event :Event) ->None
Disable the hover effect on the return button.

Method online_button_hover(self,event :Event) ->None
Add a hover effect on the return button.

Method online_button_hoverl(self,event :Event) ->None
Disable the hover effect on the return button.

10.2. Rules.py

Class Rules
Contain every Method related to the rules

Method __init__(self,canva:Canvas,root:Tk, box:Canvas, turn_player:Label,mode:int=0) -> None
Setup the Tkinter window settings.

Method quit_button_clicked(self, event:Event) -> None
Create a cross that when clicked, exit the rules window.

Method quit_button_enter(self, event:Event) -> None
Add a hover effect on the cross button.

Method quit_button_leave(self, event:Event) -> None
Disable the hover effect on the cross button.

10.3. Elements.py

Class Elements
Contain every Method related to the lobby

Method __init__(self,id:int,box_w:float,box_h:float,x:float,y:float,canva:Canvas) -> None
Register multiple information such as the player_id, coordinates, the board game size…

Function get_player(self) -> int
Return the player id.

Function get_box(self) -> tuple[int,int]
Return the box's rounded dimensions coordinates.

Function get_coords(self) -> tuple[int,int]
Return the clickable board game button coordinates.

Function get_box_size(self) -> tuple[int,int]
Return the box dimensions.

Method set_box(self,ligne:int,col:int) -> None
Register the box dimensions.

Method set_coords(self,x:int,y:int) -> None
Register the box coordinates.

Method change_player(self) -> None
Change the player turn.

Class Pawn(Elements)
Contain every Method related to the pawns and take Elements as parent class.

Method __init__(self, id :int
,canva:Canvas,box_w:float,box_h:float,x:float,y:float) -> None
Get the Elements class information.

Method draw(self) -> None
Create the pawn on the board.

Class Ring(Elements)
Contain every Method related to the rings and takes Elements as parent class.

Method __init__(self, id,canva,box_w,box_h,x,y) -> None
Get the Elements class information.

Method draw(self) -> None
Create the ring on the board.

10.4. Calculation.py

Function find_closer(x: float,y: float) -> tuple[int, int]
Search and return the closest point around specific coordinates.

Function compute_distance(coord1 : tuple[int,int],coord2 : tuple[int,int]) -> float
Return the distance between 2 different coordinates.

Function position_in_list(x: int,y: int, liste: list) -> bool
Return if the coordinates are in the list.

Function get_index(x:int,y:int,liste: list) -> int
Return the coordinates positions (x,y) in the list.

Function find_line_player(player:int,liste:list) -> int
Return the position of the player's line.

Function get_middle(x1:int,y1:int,x2:int,y2:int) -> tuple[int,int]
Return the middle of a line.

Function get_around_middle(x1:int,y1:int,x2:int,y2:int) -> tuple[tuple[int,int],tuple[int,int]]
Return the 2 points new to the middle point of a line.

Function get_line_index(x:int,y:int,x1:int,y1:int,liste:list) -> int
Return the position of the line in the list.

Function get_index_next_line(x:int,y:int,x1:int,y1:int,liste:list) -> int
Return the position of the next line in the list.

Function pos_in_line(x:int,y:int,line:list) -> bool
Return the position of the point in the line.

Function possible_position(x:int,y:int,liste:list) -> bool
Return if the coordinates are a possible position.

Function get_all_index_line(player:int,liste:list) -> list[int]
Return taken spots by a player in a line.

10.5. Game.py

Class Game
Class that handles the game, its graphics and does everything linked to the game actions or the player's input.
Contain every Method related to the game.

Method __init__(self,canva_lobby:Canvas,root:Tk,blitz_mode:bool,ai:bool) -> None
Setup the tkinter window and every class variable for the game.

Method update(self) -> None
Updates the game by calling the drawing_update Method and unbinding the click event if the game is over.

Method update_score(self,player:int) -> None:
Method that will update the score of the player in the circle on the bottom of the screen

Method move(self,event:Event) -> None:
Method that will be called when the player is moving the mouse and will call a method that will select the line hovered by the player.

Method findLine(self,hover:tuple[int,int]) -> None:
Method that will find the line hovered by the player.

Method drawing_update(self) -> None
Method that will use a bunch of methods of tkinter in order to draw the game (the rings, the pawns and the board).

Method click(self,event:Event) -> None:
Method that will be called when the player clicks on the board and will call different method depending on the situation of the game.

Method bot_turn(self) -> None:
Method that will be called in order to make the bot's turn.

Method change_line(self,event:Event) -> None:
Method that will be called when the player is right-clicking and will change the line hovered by a green line if there is another line on the same box where the mouse is.

Function selectRing(self,selected_x:int,selected_y:int) -> bool:
Function that will remove the ring selected by a player if there is a ring where the player clicks.

Method place(self, selected_x:int, selected_y:int) -> None:
Method that will place a ring on the board or a pawn on the board or move a ring depending on what the player is supposed to do.

Method possible_move(self,x : int,y : int) -> None:
Method that will update the possible move list by filling it with the possible move of the player depending on the position of the ring.

Method exchange(self,new_coords : tuple[int,int],previous_coords : tuple[int,int]) -> None:
Method that will change the player and the color of pawn between the two coordinates.

Method check_line(self,line_before:bool=False) -> None:
Method that will call another to check if a line is made and if it is, will call another method that will react depending on the gamemode.

Method verif_line(self,elem: Pawn, arr: list[Pawn]) -> None:
Method that will verify if there is a line below a pawn or on the two diagonals on its right.

Method line_blitz(self) -> None:
Method that will be called if a line is made in blitz mode.
As soon as a line is made, the method will announce the winner.

Method line_normal(self) -> None:
Method that will be called if a line is made in normal mode.
This method will check who made a line and will let him remove his line and one of his pawns.

Method remove_line(self,line:tuple[tuple[int,int],tuple[int,int]]) -> None:
Method that removes all the pawns of a line from the board.

Method change_bg(self, nb:int) -> None
Method that will change the background

Method leave_button_clicked(self, event: Event) -> None:
Method to close the tkinter window in order to ga back to the lobby.

Method leave_button_hover(self, event: Event) -> None:
Method to add a hover effect on the leave button.

Method leave_button_hoverl(self, event: Event) -> None:
Method to disable the hover effect on the leave button.

Method restart_button_clicked(self, event: Event) -> None:
Method to empty the board and set the turn counter to 0 and the player turn to 1.

Method restart_button_hover(self, event: Event) -> None:
Method to add a hover effect on the restart button.

Method restart_button_hoverl(self, event: Event) -> None:
Method to disable the hover effect on the restart button.

Method rules_button_clicked(self, event: Event) -> None:
Method to open the rules window.

Method rules_button_hover(self, event: Event) -> None:
Method to add a hover effect on the rules button.

Method rules_button_hoverl(self, event: Event) -> None:
Method to disable the hover effect on the rules button.

Method right_button_clicked(self, event: Event) -> None:
Method to change the background image.

Method right_button_hover(self, event: Event) -> None:
Method to add a hover effect on the right button.

Method right_button_hoverl(self, event: Event) -> None:
Method to disable the hover effect on the right button.

Method left_button_clicked(self, event: Event) -> None:
Method to change the background image.

Method left_button_hover(self, event: Event) -> None:
Method to add a hover effect on the left button.

Method left_button_hoverl(self, event: Event) -> None:
Method to disable the hover effect on the left button.

10.6. EndGame.py

Class EndGame
Set up a window that will appear on the Red player win.

Method __init__(self,canva:Canvas,root:Tk) -> None
Set up the tkinter window.

Method menu_button_clicked(self, event:Event) -> None
Delete the player win window.

Method menu_button_enter(self, event:Event) -> None
Add a hover effect on the menu button.

Method menu_button_leave(self, event:Event) -> None
Disable the hover effect on the menu button.

Class EndGameRed
Set up a window that will appear on the Red player win.

Method __init__(self,canva,root)) -> None
Set up the tkinter window.

Class EndGameBlue
Set up a window that will appear on the Blue player win.

Method __init__(self,canva,root) -> None
Set up the tkinter window.

Class EndGameDraw
Set up a window that will appear when the game is a draw.

Method __init__(self,canva,root) -> None
Set up the tkinter window.

10.7. Multiplayer.py

Class multiplayer
Class used in order to display the multiplayer screen

Method return_button_clicked(self, event: Event) -> None:
Method called when the return button is clicked

Method return_button_enter(self, event: Event) -> None:
Method called to enable the hover effect on the left button.

Method return_button_leave(self, event: Event) -> None:
Method called to disable the hover effect on the left button.

10.8. test2client.py

Class Client
Handle the client side of the network part.

Method __init__(self) -> None
Setup the tkinter window.

Method start(self, host='127.0.0.1', port=5000) -> None
Start looking for a server to connect to at the host and port information given.

Method start_receiving_thread(self) -> None
Set up a thread to execute the receive_messages method.

Method receive_messages(self) -> None
Check if the server has sent a message, and if the server sends a message, it displays the message.

Method send_message(self, event) -> None
Display the message sent on the client and server side.

Method display_message(self, message) -> None
Set up the tkinter window where the message will be displayed.

10.9. test2server.py

Class Server
Handle the server side of the network part.

Method __init__(self) -> None
Setup the tkinter window and create a client list.

Method start(self, host='127.0.0.1', port=5000) -> None
Start the server and listen for connections at the host and port information given.

Method start_server_thread(self) -> None
Create a thread to execute the accept_clients method.

Method accept_clients(self) -> None
Look for clients and accept them, then start a thread to execute the handle_client method.

Method handle_client(self, client_socket) -> None
Display the message sent on the client and server side and close the connection with the client if necessary.

Method broadcast(self, message, sender_socket) -> None
Encode and send the message to the client.

Method send_message(self, event) -> None
Display the message sent on the client and server side.

Method display_message(self, message) -> None
Set up the tkinter window where the message will be displayed.