Université de technologie Belfort-Montbéliard

Cycle préparatoire : TC04

Rapport de projet LP2A

Sujet :

Réalisation du Jeu LUDO King en JAVA

Réalisé par :

AMRI Nabil

&

ALBUZLU Gökdeniz

Semestre Printemps-2021

Sommaire

Introduction:

In this you will find how we developed the Ludo King game. This game was implemented in JAVA because it needs an object-oriented language to develop it. You will find step by step all the choices we made to create it. First, we will speak about the code and its structure, then how we created and organised the interface and finally we will evoke the difficulties encountered.

To develop the Ludo King game, we decided to propose 2 different modes of game, a Solo mod in which you must beat 3 CPU Players. And a multiplayer mod which can be played with 4 players. Before to start the report, we want to say that our program is long because we tried to propose a clean graphical interface to the user.

A) Welcome Menu.

When we open the program, the first screen that we meet is a simple welcome menu as in the other games. The player has 4 possibilities, he can launch a solo game, a multiplayer game, he can check the game rules if he does not know them well. And he can close the programme.

B) Solo Mod.

The Solo version is the first version that we developed. Indeed, we decided to play always with the yellow player in solo mod. Also, the CPU players are Red, Green and Blue players. The Solo Game was a little bit difficult because we had to create an artificial intelligence which is capable to challenge the players. In this part we will explain our game step by step.

The Throw Step:

First, we created a JButton called "ThrowButton" to propose a dice interface to the player. To organize the Dice Throw we created a "ThrowDice.java" Class in static in order to use it also in the multiplayer version. This method is a simple method it takes in parameters the pictures, design labels, and a random number which allow us to throw dice. In this function we used a big switch statement with all the cases (if he throws 1, if he throws 2…). This part is long because we care about the graphical user interface, indeed you can see that we are setting each time the position of different image of dice and the different labels to give a better interface to the player. Then, when throwing Dice is finish for the player, An Ok Button will appear. This button will generate the dice of the CPU players. And at the end of the throwing round, a continue button will appear to clean the interface. Indeed, we set the dice pictures to their initial position (make them invisible) and we continue like that for throwing Dice.

Throwing 6:

As you know, you must throw a 6 to get out the Square and to begin to play. Indeed, when the player throws 6, we propose him to put one of his 4 Pions out the Square. Consequently, we created a "InsideGameRules.java" class to verify some conditions for instance, if he throws a 6, he continues to play. It is the same case for the CPU players. And we block (we cannot click them) our Pions if it is not our turn, it is logical.

Moving a pion for the player:

After throwing a 6 we can finally start to play with our pion. To organize the moves correctly. We decided to create a "Coordinates.java" and "Lists.java" Classes. The lists classes contain the coordinates (depending on x, y, width, and height) of all the positions. The coordinates class contain the getter and setters for the previous class. At the end of the Lists class, we obtain an array for each colour with their positions. Moreover, we created a Game variables class in which we will find useful static variables (like positions, random number, pions counter), Indeed, we create 16 variables in static for positions. One for each pion. By doing this, we will attribute an array for pion. And we will simply increment their position variable depending on their throw to move them.

Moving a pion for the CPU player.

Here, we are using the same methods for the CPU players with Lists class. However, The CPU method is different from the player moving method. Because we propose some additional functionalities. Indeed, we created a class AIForPlayers.java in which there are 3 static method for each CPU players. They firstly verify if the CPU has a 6. Then, they will move their first pion until that they reach the final case. Even if the CPU player throws 6, they will not move out a second pion because this is risky (in terms of eat). When the first pion of the CPU reaches the home case. They can directly move a pion without throwing 6. By doing this, we propose a real difficult challenge to the player. To organize this functionality, we are just counting the numbers of pions out of the square for each player by using the static variables defined in Game variables class.

Eating pion

As you know, in this game we have the possibility to eat the pions of other colours (they can also eat us). To develop this, we created a “Eat.java” class in static. In this class we developed 4 static methods (yellow, red, green, and blue). In which we are using a lot of if statement. At the first appear, it can seem like ugly, but we do not have the choice, we must compare all the possibility (For instance, if the first pion of blue eats a third pion of yellow….). When a pion is eaten by another pion, we decided to send it at the first position of his colour. Indeed, they will not have to throw again a 6 to put it out the square. We decided this functionality to accelerate the game and make it funnier. Because as you know the game is already very long. As a consequent when a pion is eaten, his position variable is initialized to 0.

Winner

Finally, the last step of the game is to find the winner. As you know you can win only if your 4 pions reach the home case. Indeed, we decided to create a “winners.java” class in static. In which we have 4 methods in order to verify if one of the players has won the game. If one of the players win the game, we display a WIN message, and we close the game automatically after the message.

C) Multiplayers Mod.

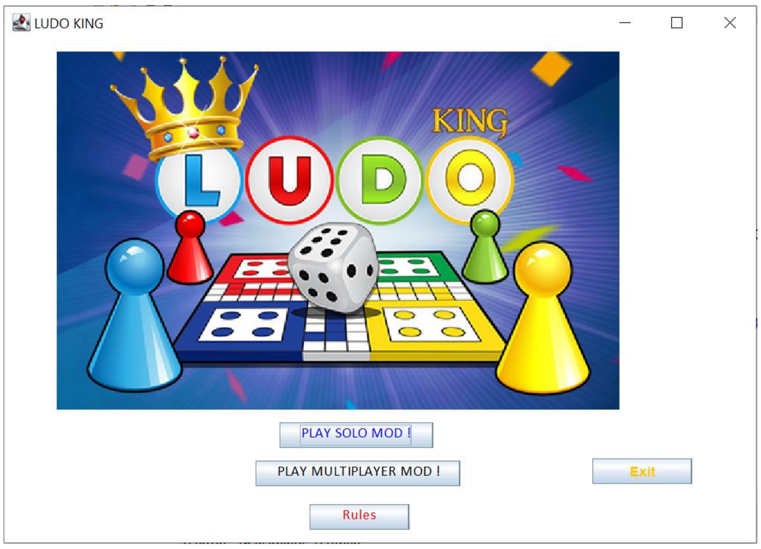
This mod is very similar to the Solo Mod without the CPU players that is why we will not detail it so much. There are 4 additional Throw Button, One for each colour. We are using here the same functions and the same methods to throw Dice, to handle the eating and for the GUI. At the end we verify the winners by using the same function.

INTERFACE

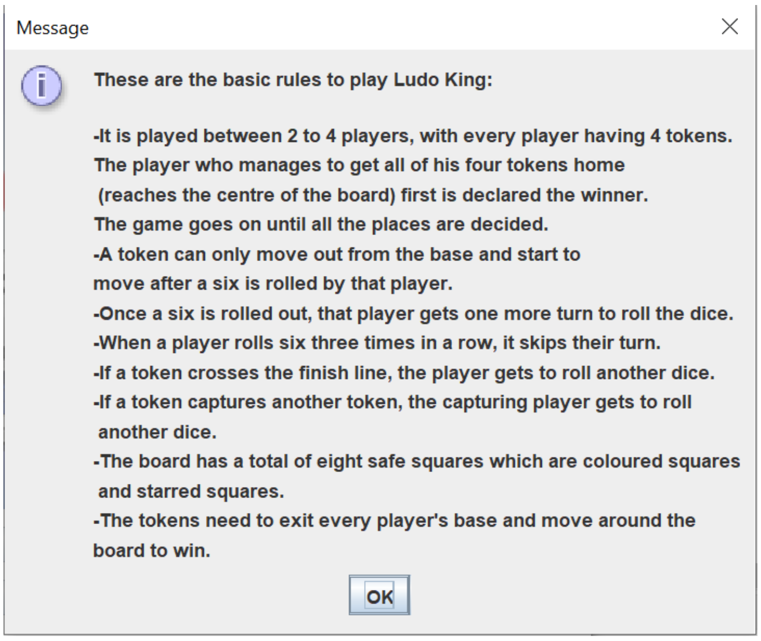
In this part we will speak the organization of the interface and the different choices we decided to implement it.

Menu window

When the user starts the game, the first window is the menu. Several choices are proposed to the user: solo mode, multi-mode with 4 players, consults the rules of the game and exit the game. All these possibilities are JButton. The picture is setting in a JLabel.



The player can consult the rules of the Ludo KING games in clicking on “Rules”. A Messagebox appears with the details of the rules. We decided to put it in a Messagebox so that the user can close it and return to the menu without having to quit and relaunch the whole game.


Message

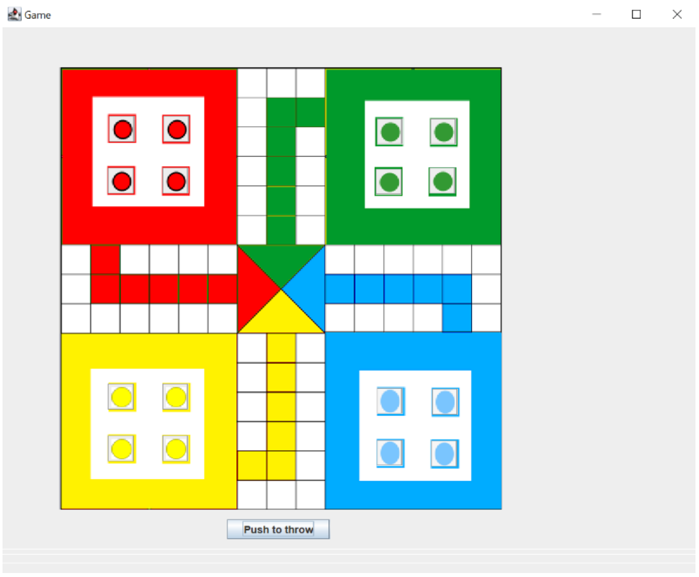
These are the basic rules to play Ludo King:

-It is played between 2 to 4 players, with every player having 4 tokens.
The player who manages to get all of his four tokens home
(reaches the centre of the board) first is declared the winner.
The game goes on until all the places are decided.
-A token can only move out from the base and start to
move after a six is rolled by that player.
-Once a six is rolled out, that player gets one more turn to roll the dice.
-When a player rolls six three times in a row, it skips their turn.
-If a token crosses the finish line, the player gets to roll another dice.
-If a token captures another token, the capturing player gets to roll
another dice.
-The board has a total of eight safe squares which are coloured squares
and starred squares.
-The tokens need to exit every player's base and move around the
board to win.

OK

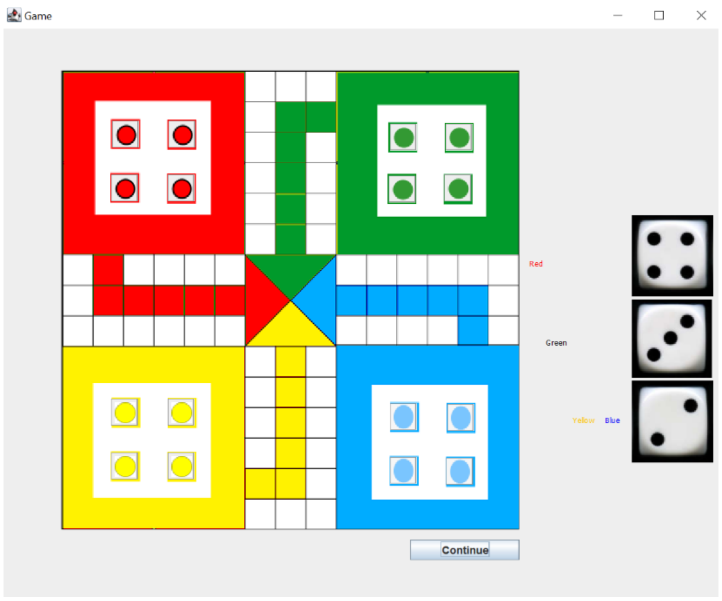
Solo mod:

When the user will choose one of the play modes, another frame will automatically appear. The game take place in this well-organized frame where we find the game board, the different face of the dice, the different pieces, and the different commands.

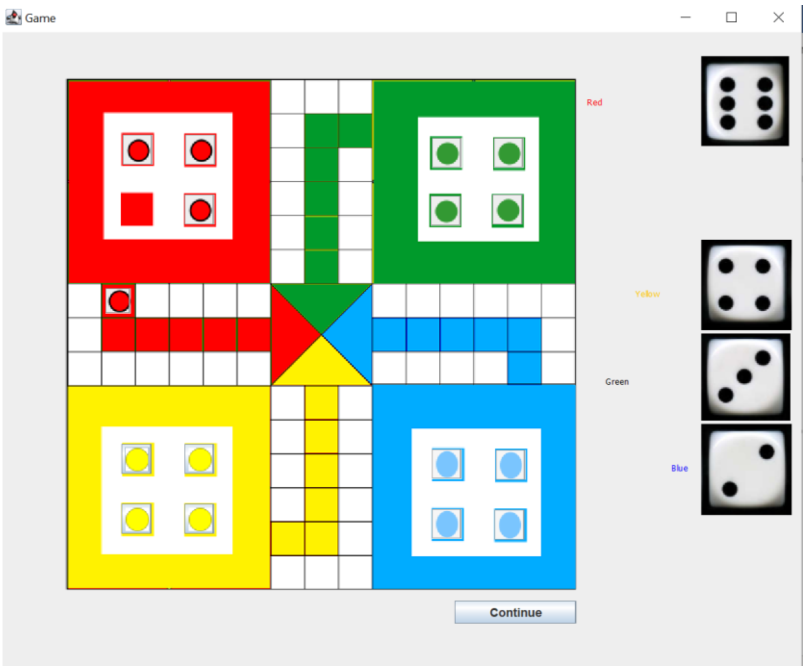


The sixteen pieces are implemented with JButton and the different Image are setting with the function "set.Icon()" . Furthermore, the game board is setting in a Jlabel with the function "set.Icon()" too.

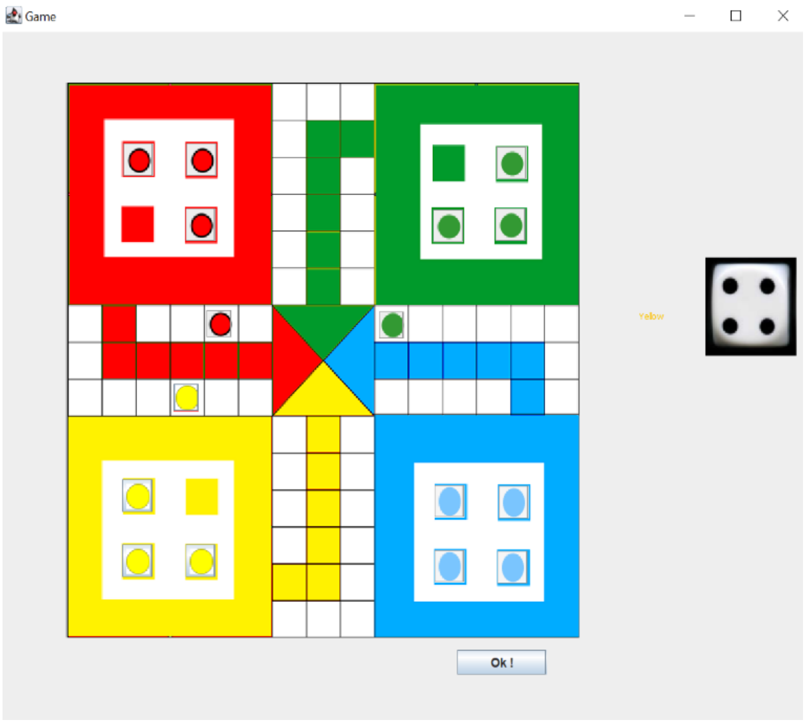
We decided that the player begins to throw the dice. In fact, the user pushes the JButton "push to throw "and one face of the dice will appear. Then the dice will be automatically thrown for the other player (AI). After that, the color of each player will be affected to a face of the dice with a JLabel for each color.



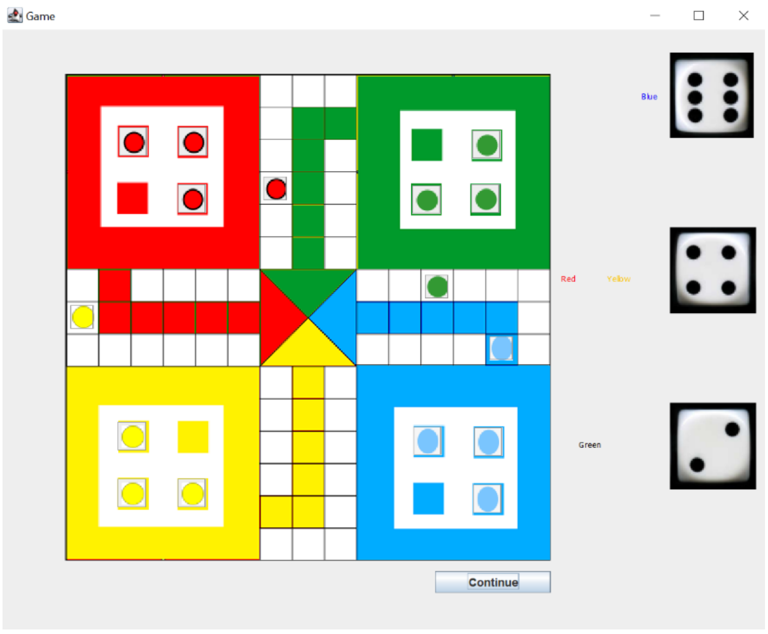
At the beginning, a player must have a six in order to move one of his pieces. Therefore, in this mode (the solo mode), the pieces of the player can't be moved no matter the number if it is not a six. Thanks to the JButton "Continue", the player access to the next turn and thrown another time the dice. When the turn is finished the different faces of the dice will be invisible with the function “setVisible()”.



In this case, the red AI get a six so a red peace is positioned on the start case with the function “setBounds()” (we will use this function during the whole game in order to move the pieces by changing the coordinates). When the player gets a six, he must select the piece to move, and he can bring in the game another piece if he gets a six. For the AI, it is a little beat different because we made a function which distinguish two cases if it gets a six: the AI could bring in the game another piece but if he can eat another player’s piece by verification the coordinates, it will eat the piece.



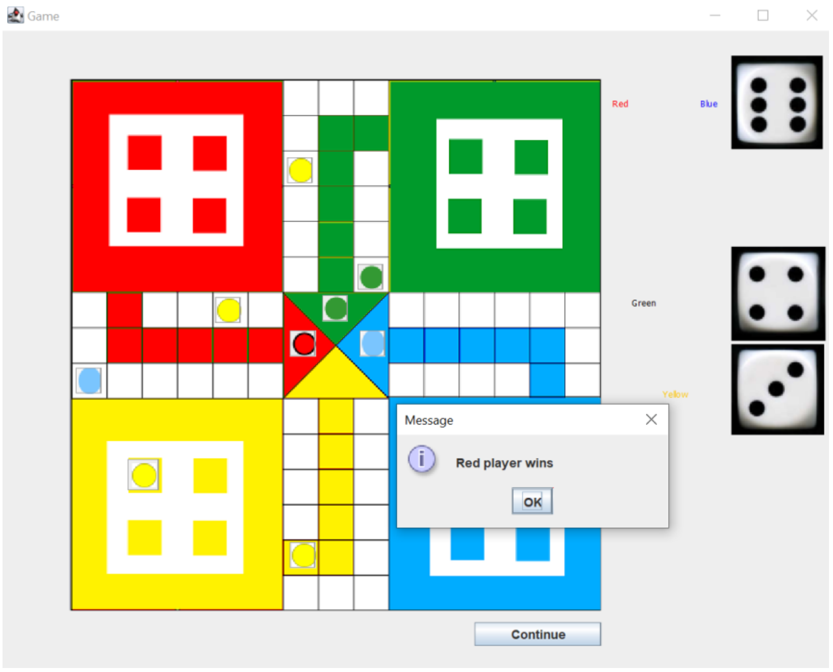
The user throws the dice and gets four and select the piece to move in the game.



The yellow piece progress for four cases. The blue player gets a six and bring in the game a piece.

Each player continues to move his pieces to the final case in order to finish the game.

Finally, when all the pieces of a same color are in the finish case, the player win, and a message appear with his color in a MessageBox. The player can exit the game by pushing the button "OK".

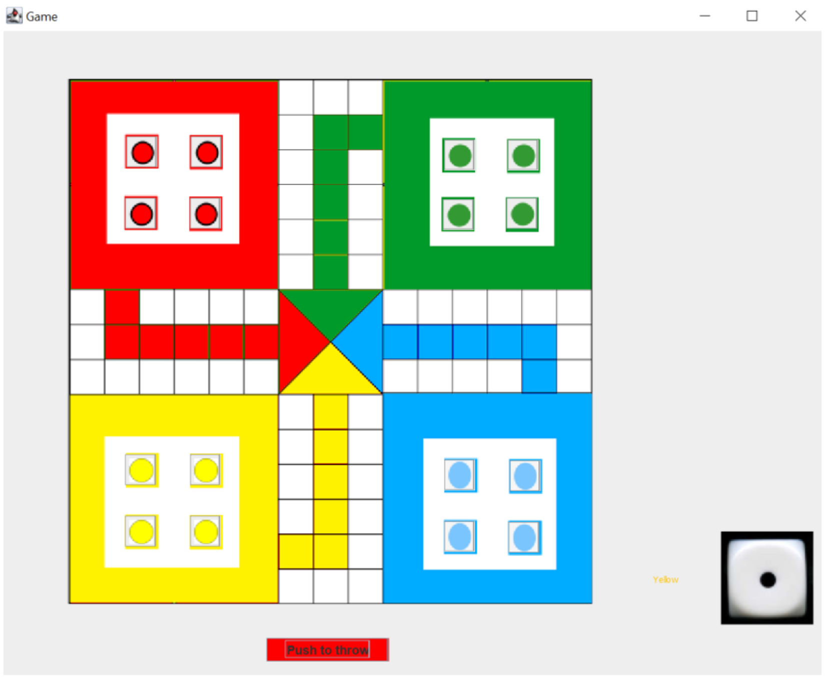


Red player wins

Here all the red pieces are in the finish case so the red player wins.

Multiplayer mod

In the multiplayer mod, it is the same principle except that each player will throw the dice and play independently. They can bring a piece in the game by pushing it or eat another piece when it is possible. The JButton take the color of the player who must throw the dice.



The Button is red so the red player can throw the dice.

Difficulties encountered.

First difficulty :

At the beginning we had a problem with the display of each pieces. In fact, we couldn't display all the pieces at the same time and keep them visible until the mouse over. However, after research we solved this problem with the functions Validate () and Repaint (). It was a related with the refresh of GUI.

Second difficulty:

When the AI throw the dice and get a six, it will bring in the game another piece, but we wanted a smart AI. In fact, we made a function "Eat" which check if the AI can eat a player if it moves for six cases and not bring another piece in the game. This function took time to develop it.

Conclusion

This project was very enriching and useful. Indeed, we learn more about the language JAVA and especially the manage of the interface. Moreover, it challenges us to optimize to code and not copy/paste in order to reuse the code but to find a way to minimize the number of codes' lines. Finally, we are very happy for our Java Program.