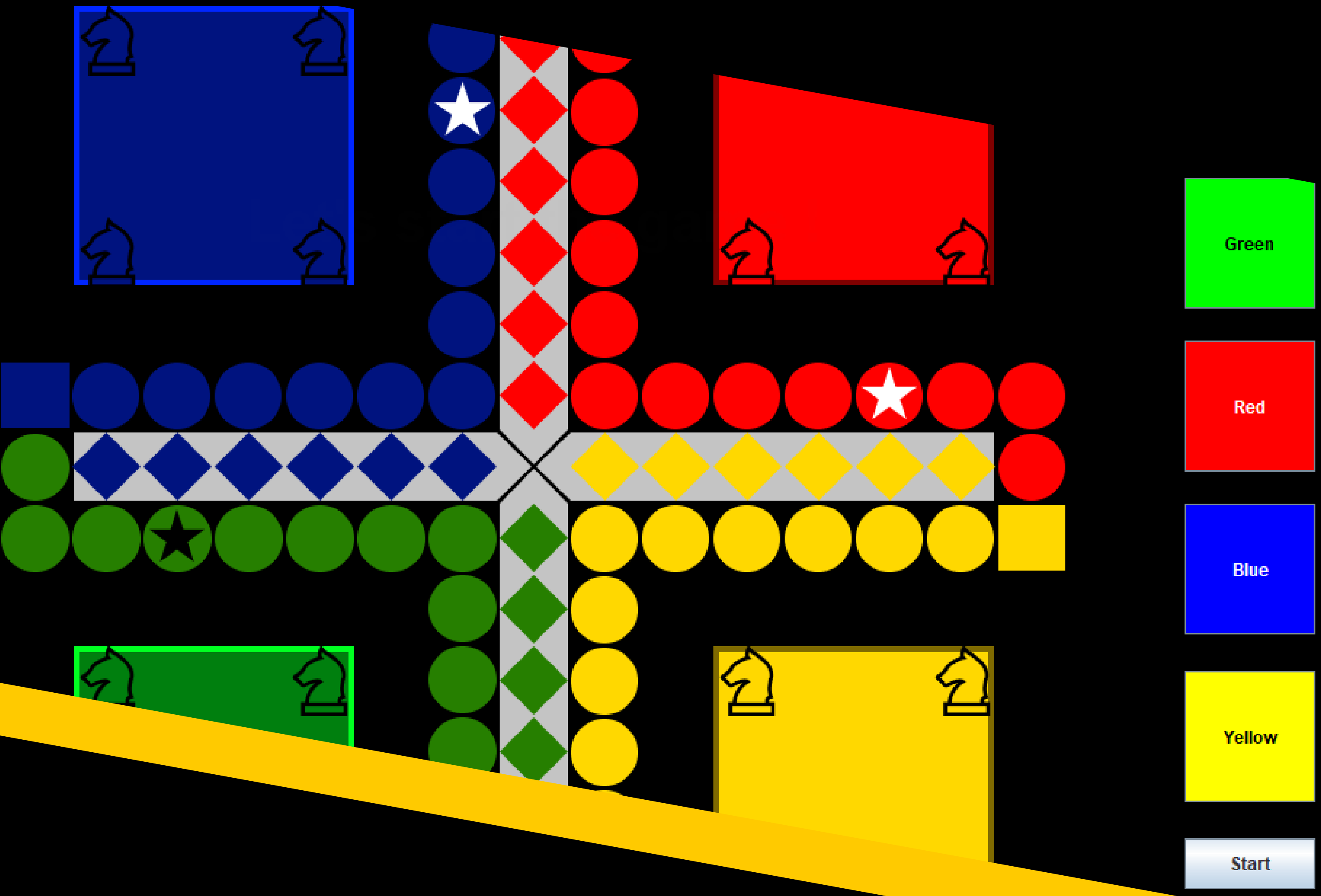
Project report LP2A

GAME OF LUDO



LIGNON Thomas and VIALA Alexandre

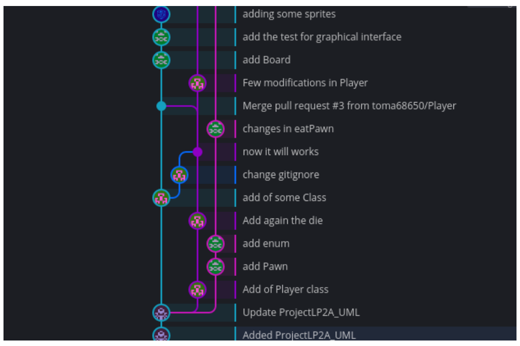
Project's report

I. Introduction

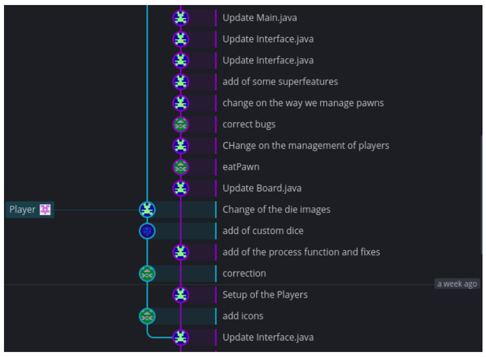
During our study path, we were asked to program a ludo game in java language. This game is played by the intermediary of a JFrame, and we made 4 players playing against each other.
The ludo game origins come from India during the 6th century. The game as we know it, was imported in England in 1896.
As we thought first, the Ludo game isn't the same game as the horse game. In the ludo game, the path the pawns have to get through is a little bit smaller.

II. The way we organized the work

When the subject of the project was announced, the first thing we did was to download the game "Ludo King" as our teacher advised us. Thanks to it, we learned the rules really quickly.
Then we thought about all the classes, functions and links between our classes in our program. For this project, both of us were using the IDE eclipse and gitHub desktop. We had some issues with GitHub at the beginning but when we got rid of them, we really began the programming. We decided to create the basic elements, so Alexandre was in charge of the player and the case while Thomas was in charge of the pawns and the board.We also made the first methods of these objects and created the die. Then began the hardest and longest part of this project: the graphical interface and its link with the rest of th programm.
Then we decided that Alexandre would focus on the interaction with the player (like selecting/moving the pawns) and Thomas would create all features for the graphical interface.



adding some sprites
add the test for graphical interface
add Board
Few modifications in Player
Merge pull request #3 from toma68650/Player
changes in eatPawn
now it will works
change gitignore
add of some Class
Add again the die
add enum
add Pawn
Add of Player class
Update ProjectLP2A_UML
Added ProjectLP2A_UML

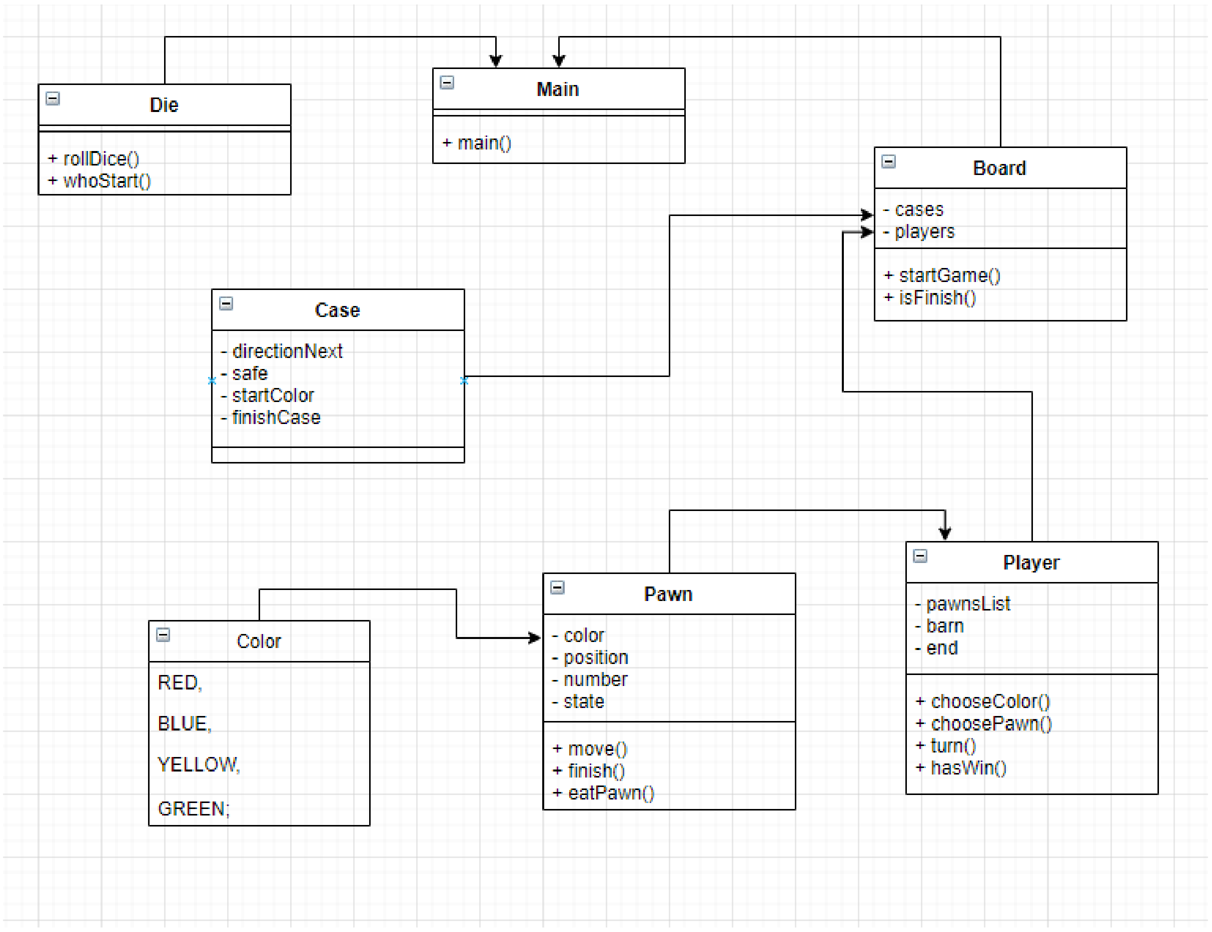


Update Main.java
Update Interface.java
Update Interface.java
add of some superfeatures
change on the way we manage pawns
correct bugs
CHange on the management of players
eatPawn
Update Board.java
Player
Change of the die images
add of custom dice
add of the process function and fixes
correction
a week ago
Setup of the Players
add icons
Update Interface.java

III. The organisation of the code

For Object-Oriented Programming, it’s better, and we were advised, to prepare an UML diagram. But the more we were programming the more we derived from our initial diagram. So here we show you the differences between our initial diagram and our final one.

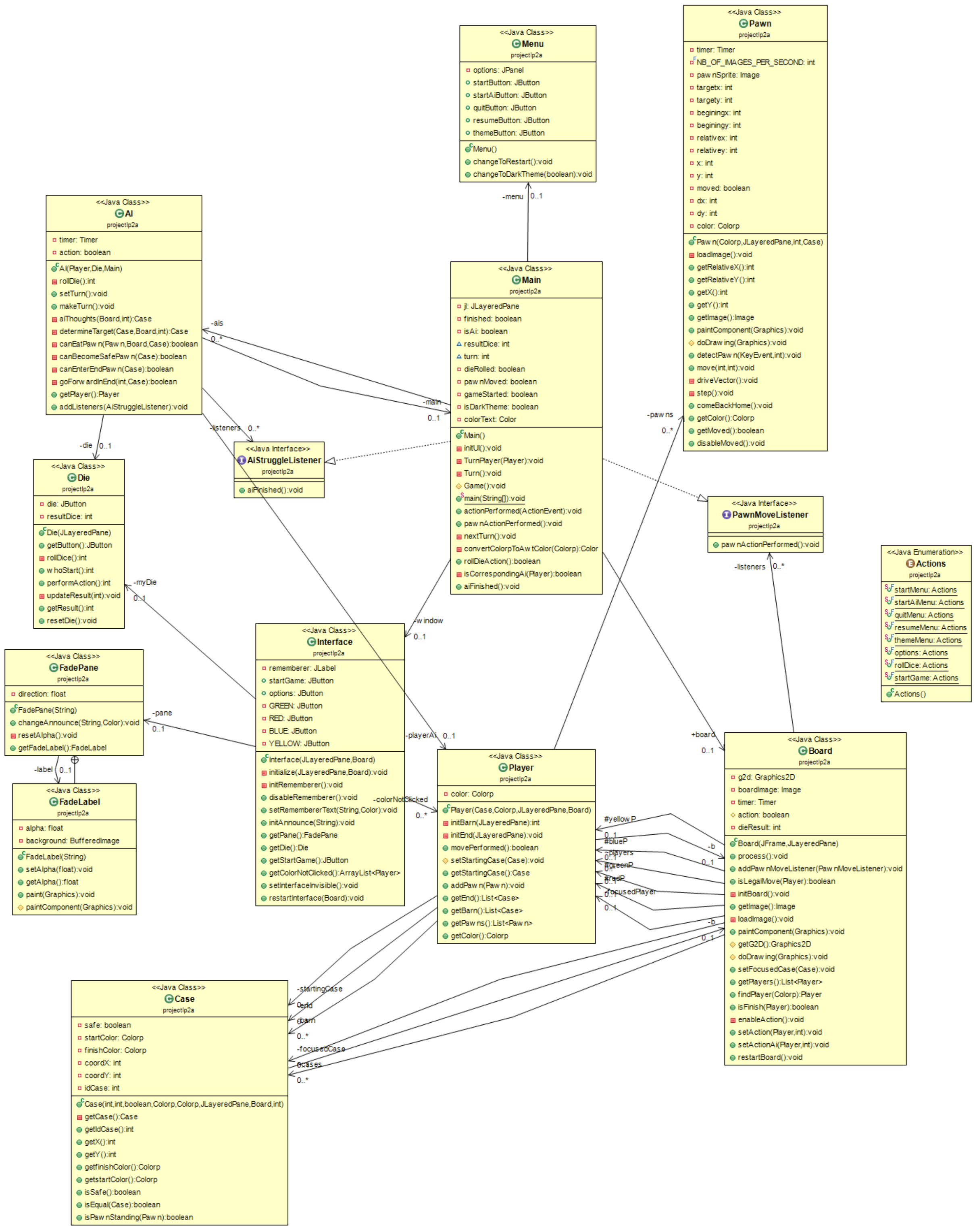
The first diagram



The final diagram

We can see there are a lot more things in this second diagram than the initial one. This content increase is due to all the things we omitted and the one we could not think of. By example, we had omitted all the things linked with the Swing interface.

We also decided to implement a menu and an interface to make the game easier to play with. And we added some interfaces to simulate a java listener. To manage the information of the buttons in the same file (the main file), we also added an enum : Actions, which define several actions : approximately 1 by button.



This diagram has been generated thanks to the Eclipse plugin ObjectAid

IV. The structure of the code

The game starts with a menu where the player chooses to start a game or to quit.

Player

In the player we created 3 lists: - a list of barn, the barn is where all pawns of the same color begin the game.

- a list of endcases, at first we didn't know how we should define them, because some rules say it's in the middle of the board and some say that it can be on the middle lines that meet in the middle. At last we made the endcases on a line and not on a single case.
- a list of pawns. This list contains all the pawns for each color. It helps us when we call movePerformed(), that does a for each loop to verify if a pawn has moved or not to avoid that someone skips his turn or plays more than once.

Pawn

When we look at the beginning of the class Pawn, we can be troubled with the different coordinates but they aren't that complicated to understand.
First we have the relativex and relativey. Because using the x and y seemed a little bit tiring and will get the code more complex to read, we decided to give a number to the cases, so we could know precisely where a pawn is located. Also all the coordinates in this function are relatives so we can find ourselves easily.
Next, the beginingx and beginingy are the coordinates of a pawn when in the barn, these coordinates are the same during all the game.
And so we have the target coordinates, these are also relative but they designate the coordinates where the pawn is moving to.

In this class we also manage the movement of the pawns thanks to vectors.
The step function is the function that makes the pawns movement animated.
The timer that we use is to refresh the sprite of the pawn that is moving.

Case

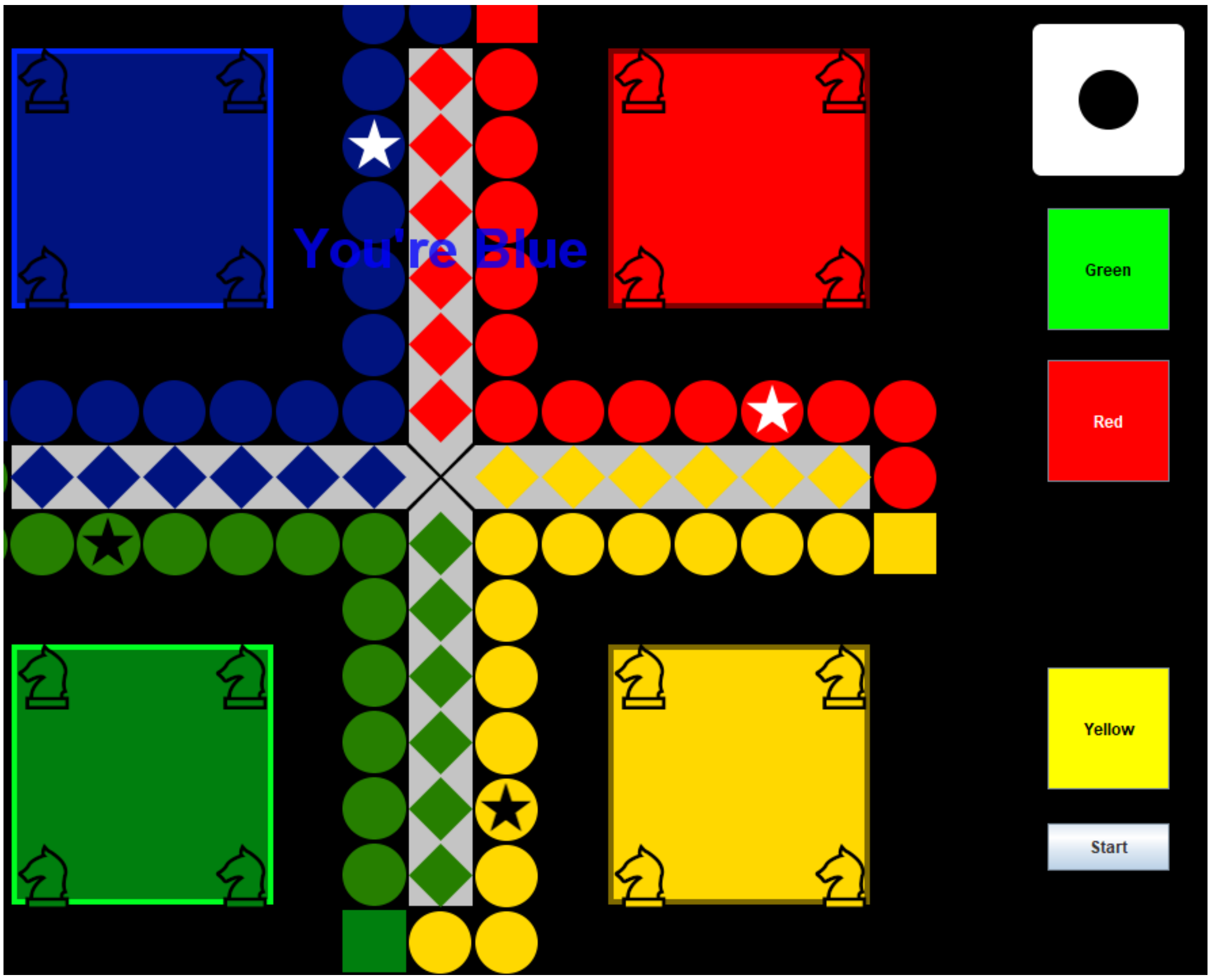
A case is defined by quite a lot of arguments. We have the coordinates of the case, a boolean that tells if the case is safe or not. We also verify if the case is a start case or a finish case (it's null if not). We also ask an integer so the case will get a number which is easier for us to find the case we want.
The start case of each color is defined in the class Board.

Board

This is the place where all the methods and attributes linked to the movements are made.

The board is linked with a Swing Timer. This timer makes an action every millisecond : it check if a human player is currently playing, if it is the case, a big method, process() is called.
The process() method is one of the most important of the game. Actually, it will check if the action asked by the player by rolling a die and then clicking on a case can be performed. If this movement is a legal move then the process() method will move the pawn pointed. If it is not the case, the process() method will do nothing. The main will be notified at the end of the function and if no horse moved after pressing on the case, then the main will print an error to the player. If a horse moved then it will notify the main to go on and do the next operation. This operation is whether waiting for the player to do a new action if he made a 6 or skip to the next player if he did not make a 6.

Interface
It’s in this class that all the graphical interface is created except for the “launch die”. First, you will find the buttons that the players will click to choose their color, we also have decided to show a FadePane to tell the player which color he chose. Once you have clicked one it will disappear to avoid any problem. The order the buttons are clicked define the order the players play.

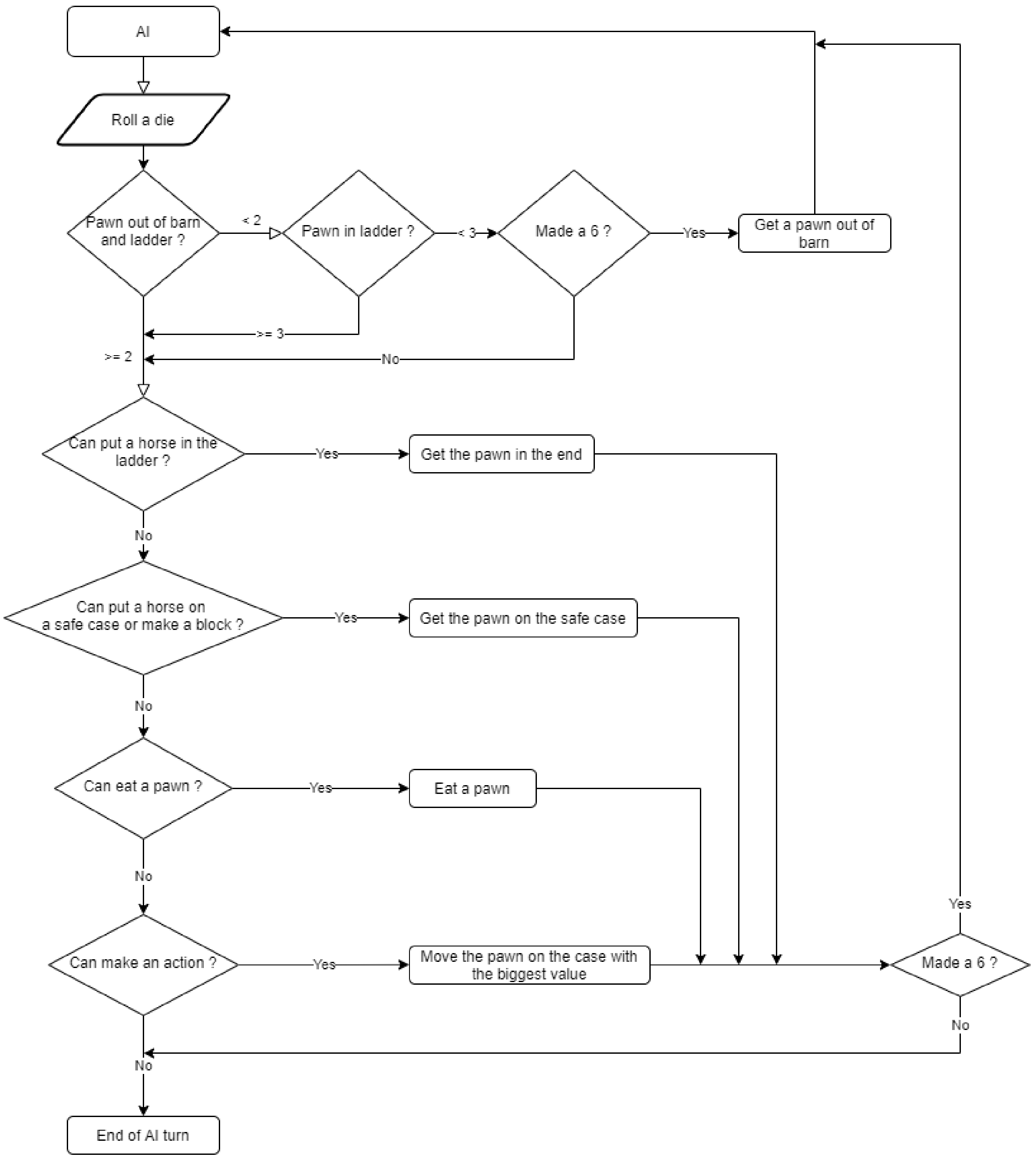


Next, we defined the button that launches a die. This one isn't written in the class Interface but in the class Die, because it always returned an error when we clicked on it. The action of this button isn't written, it's initial picture is the face 1 of a die. When you click on it, the result will turn gray, so that players can't click on it if they haven't moved a pawn.

We also created an option button that returns to the menu whenever the player wants.

AI

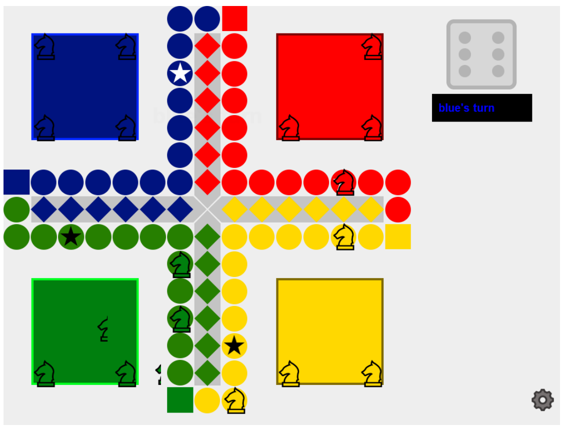
The AI was an optional thing to add. We made her think with few options. The way she thinks is described just below on the algorithm. The AI uses the process() method directly without using the timer of the board because, they also have a timer which cannot be easily synchronized with the board timer. They play at a decent speed to understand more or less what move they made without being too confusing. We added a button on the menu to choose to start a game with or without ai.

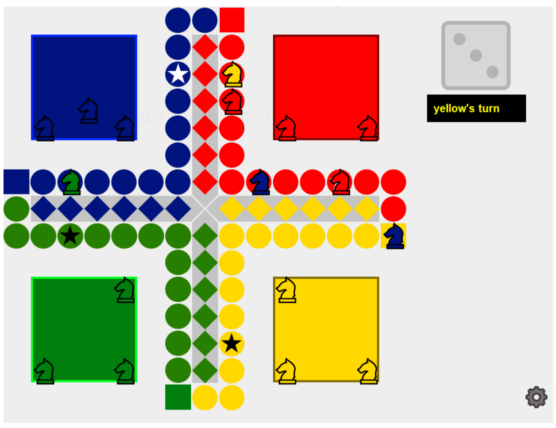


V. Ways to improve our code and improvement made

Bugs corrected

During our project we meet several errors. By example we had some graphical bugs that happen oftenly. It was due to the repaint method of the Swing library which presents bugs when one does not set up everything correctly.





We met other bugs on the AI mainly when we wanted to make a turn.
In java, it is difficult to use a java loop to build a gameplay loop, so one needed to use actionListener to detect an action and then start another action just after. It was then difficult to solve a bug when we did not know where to find each time. We succeeded by finding the most important bugs and correcting them. We probably misunderstood something could have helped us to make the code easier to understand and to debug.

Improving our code

At the end of our project, we realised, after checking every rule we introduced, that some were lacking, like when you eat a pawn, you can play again.
We tried to make all the movements of the pawn the same, with same speed, trajectories almost similar, but after some attempts we decided that it was not the most important. The best would have been to make the movements a bit faster.
Also, our menu is quite austere, we can improve it by adding colors and maybe an option part where we can choose the numbers of players we want, or the difficulty of the AI.

VI. Conclusion

Thanks to this project, we both learned a lot in Object-Oriented Programming and in java. Even if the rules of this game were pretty easy to understand, it was more complex to implement.
During this project, we manage to implement all the main features. Though, we have smooth-moving pawns with all the basic rules of the ludo. One succeeded in implementing Artificial intelligences with basic reasoning. We also put a little menu in our project to have a more user-friendly interface.

VII. Link of our github
github.com/toma68650/ProjectLLP2A