CS1006 Royal Game of Ur Practical

200014517/200001541/ 200015381/200013165

Tutor: Ognjen Arandelovic

28 Apr 2021

Summary of Functionality

Our program supports the following functionalities:

- Able to provide the user with a menu that allows the user to:
    - Select within the three different types of game requested by the specification:
        - Playing against a local human player
        - Playing against an AI opponent
        - Taking part in a network game against another human player
    - start a new game whenever they want.
    - quit current game.
- Able to provide a graphical representation of the game displaying:
    - a 3x8 grid representing the game's board.
    - stones of distinct colors (green and red)
    - the number of stones of each color in the supply
    - the number of stones of each color that have been moved off the board
    - the five rosettes situated in their specified places and represented by a special drawing.
    - the dice (the rightmost cell from each of the three rows)
    - the outcome of each die (0 or 1) after rolling the dice.
    - the menu bar previously described containing the menu and the button for rolling the dice
    - simple and intuitive animations when rolling the dice or moving a stone on the board
    - an additional window used for retrieving input from the user.
    - an introductory window that welcomes the user and presents a brief set of instructions.
    - different messages that signal the user of different events that took place such as:
        - one of the players has won.
        - one of the players involved in a network game has left the game.
        - other messages that refer to issues when trying to create or join a network game.
- When playing the game, the program allows the user to:
    - roll the three dice by clicking the "Roll Dice" button
    - move a stone by clicking it.
    - bear a new stone onto the board by clicking the round button that represents all the stones the supply of stones.
    - bear off a stone that is near the end of its path.
    - choose any play mode they want from the menu.
    - start a new game or quit the current one also from the menu provided.
- Able to provide a MVC architecture within the program that offers a clear separation of concerns.
- Able to provide a simple and clear set of rules consisting of the following:

- each player has 7 stones (most of the games that we found also assigned each user the same number of stones).
- the first player will always move the red stones.
- the player does not have to roll an exact number to bear a stone.
- each player will roll three dice to receive a number between 0 and 3 of steps they will have to move a stone.
- if a stone lands on a rosette it will result in that player rolling the dice again.
- stones of the same colour cannot share the same space, but a stone can land on another stone of the other colour to capture it (i.e. send it back to the start).
- the stones are following a usual path that can be found in most of the Royal Game of Ur out there (i.e. the path described within the practical's specification and represented in Figure 2).
- the first player to bear off all their stones is the winner.

Design

Code Design – MVC Architecture

As mentioned in the section above, our program features a MVC architecture that offers a clear separation between the game's GUI and its internals. Our program is split into multiple classes:

- **GameWindow** – represents the main GUI component of the game.
- **Game** – represents the main class used for storing the game's logic and the data about the board's state at any point during the game.
- **Controller** – represents the class that takes the user input and deals with it by manipulating the game's data.
- The rest of the classes are auxiliary to the three ones previously described.

In most cases, the way the program will operate is that, firstly, it will take the user input (e.g. clicking a stone, clicking the "Roll Dice" button etc.) via the **Controller** class and will perform a certain manipulation on the data stored inside the game's internals (usually with the help of the methods inside **Game** class). Then, the changes just made will trigger the GUI components of the program to update the game's view. This is done by having a **GameWindow** attribute inside the **Game** class, which will be used to call update operations every time a change is made on the board so that the view matches the game's internals. Then, the game is once again ready to accept input from the user, repeating the cycle until the user decides to close the game.

It is important to be noted that the update operations called on the **GameWindow** attribute from the **Game** class only access and query the data stored within the game's internals, they do not

perform any changes to it. This latter aspect offers a clear separation of concerns within the program.

Game Engine

Game Logic

We decided to represent the board by using a 2D array of integers. Each entry of the 2D array could take four different values: 1 and 2 represent tiles containing stones of a particular colour (red and green respectively), 0 for if the tile is empty and -1 for the tiles that are invalid (there are four invalid tiles: two on the first row and the two on the third row). The fact that the board's dimensions would remain constant the entire time was the main reason for our decision to work with 2D arrays of integers. Another feature worth mentioning is the fact that the player moving the red stones is always considered to be the first player.

AI Opponent

Regarding the implementation of the AI Opponent, we decided to make use of the expectiminimax algorithm. This has greatly improved the program responses to the user moves, by assigning different values to various possible outcomes and selecting the most optimal among them. After rolling the dice, the AI Opponent takes into consideration not only the possible moves that it can make, but for each move it can make, the possible moves the user might in turn make on their next turn. All this data is put through an intensive process of analysis which is filtered to obtain the best possible outcome. Because the entire process is almost instant, we decided to add 300 milliseconds delay before the AI opponent's move is performed so that the user will be able to acknowledge what dice roll its opponent had.

Network game

The implementation of the network game relies solely on the Client, ClientHandler, Server and NetworkConstructor classes. Its structure follows the same design as the one provided by both the BasicClientServerExample found in the Example folder on Studres (this folder can be accessed by following the link: https://studres.cs.st-andrews.ac.uk/CS1006/Examples/) and the example described in a YouTube tutorial that can be accessed by following the link provided in the fourth reference included in this report. The entire network connection is constructed via the NetworkConstructor class, after which the ClientHandler and Client objects will do all the work for sending messages between the two players. In our view, this implementation offers a solid network infrastructure that is also capable of dealing gracefully with unusual situations. Another example of a network game that we came across while doing research on this subject is the one provided by the link from the fifth reference. Even though we have not really used it, we decided

it at least was worth including since it played a vital role in our understanding of a network game's implementation in our incipient phase of developing it.

Graphical User Interface

GameWindow

The graphical representation of the game has been constructed with the idea of offering a simple, consistent and intuitive user experience, which makes use of the window's space efficiently by trying to reduce the redundancy factor as much as possible. One feature that highlights this latter aspect is the design decision of having only one set of three dice that fits perfectly in the right-hand side of the game's board. The stones have two distinct colours: red and green, which make them easily distinguishable, while the rosettes are represented by an image that we thought to be close to the ones drawn on actual physical **"Royal Game of Ur"** boards (the rosette drawing can be found by following the link from the third reference; the sixth reference has been used for reading the rosette drawing and storing it into an Image object, while the first reference has been used for resizing the image so that it fits perfectly inside the tile). For the sake of efficient use of space, the title bar of the game has multiple functionalities other than displaying the game's title, which depend on the play mode the user has selected. These functionalities can range from being an indicator to which player's turn it is to displaying an appropriate message when, over a network connection, the user is waiting for the other player to join or to make a move.

Board Game

The game's board is represented via the GameDisplay class, which extends the JPanel class and features a GridLayout. Each tile corresponds to another JPanel object that contains a RoundButton object, except for the rightmost tiles from each row, which are reserved for representing the three dice. These RoundButton objects have an essential role in displaying the stones on the board and creating all the move animations (i.e. by changing their colours and by setting their visibility off or on). To be able to properly shape the buttons in the form of a circle we have made use of code from the website provided by the second reference added below.

We opted for this kind of representation of the board game because it was easily compatible with the 2D array of integers used for representing it within the game's internals.

Game's Menu

For the game's menu we decided to make use of a JMenu object that would contain a button group of JRadioButtons representing the game's different play modes and two other JMenuItems

for allowing the user to start a new game or quit the current one. This implementation offered an easy, elegant, and intuitive user experience.

JDialog

Another important feature has been the frequent use of JDialog due to the benefits that they brought. Whenever an instance of the IntroMessageWindow, InputWindow and MessageWindow classes were created, a new window would show up, which would allow for the GameWindow to be accessed only after its closing. This forced the user to read and acknowledge the feedback or message that was transmitted, before being able to perform anything else on the game's board. Another tweak that is felt mostly within the program's implementation is the fact that the code written after the line that initialises a new JDialog will be executed only after the disposal of the JDialog instance. This in turn brought several advantages to the program's implementation.

Look and Feel

Another important design decision was to invoke the UIManager to set the Look and Feel. This helped us avoid issues such as the round buttons representing the stones having varied sizes on devices running on different OS, which would have caused inconsistency problems.

User Input

Controller class

For the sake of achieving a proper MVC architecture, all the user input is put through the Controller class. By implementing the ActionListener interface, the logic inside this class can properly recognise the type of the user input and decide how to deal with it (i.e. send requests to other classes to manipulate the game's data, dispose of a MessageWindow, display an animation etc).

Bugs and Defects

One defect that the program has is the inability to resize the game's window. Otherwise, if the user would have been able to resize the game's window, two key issues might have appeared: either the rosette drawings would not resize and the board would have an inconsistent look or they would have resized every time the GameWindow object would have refresh, which would have affected the game performance severely.

Another aspect of our program that might be regarded as a defect is the fact that after rolling the dice, if the user must move zero steps, they will have to click on a stone even though no change will happen to the board.

Screenshots

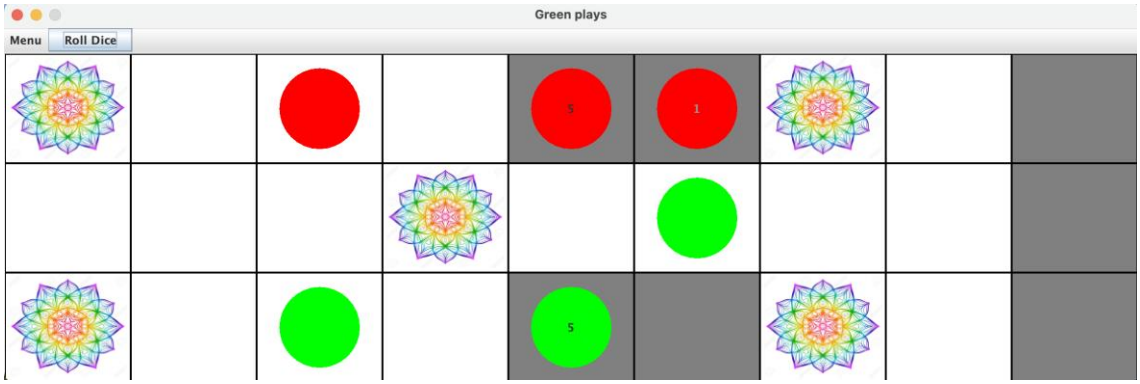

Figure 1: Representing the board

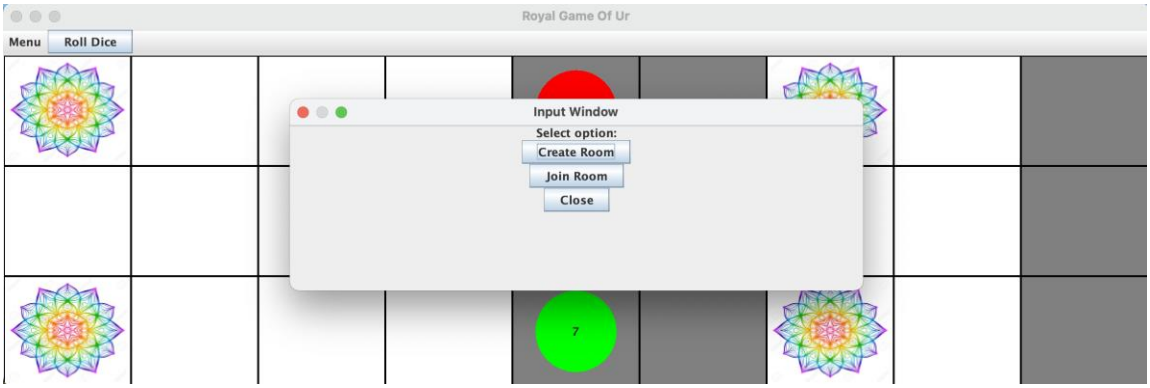

Figure 2: Representing the InputWindow before selecting to create a network game or to join another one

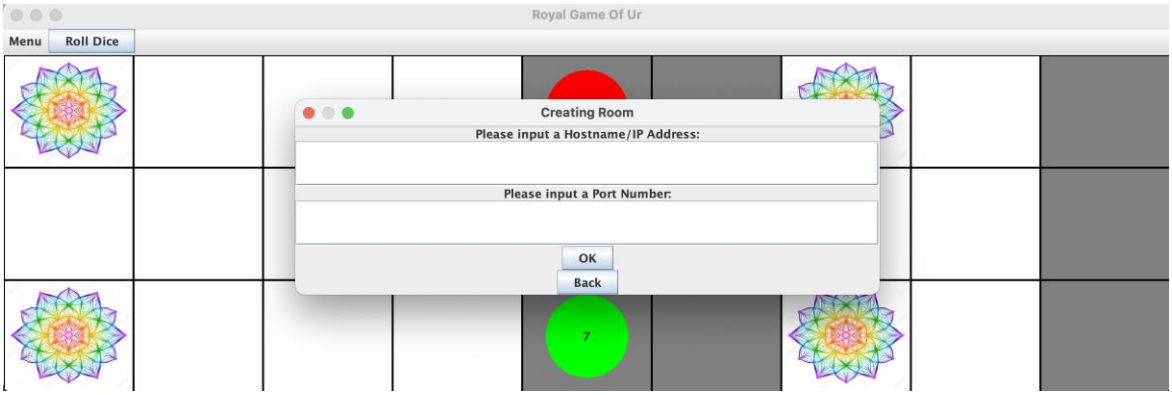

Figure 3: Showing the InputWindow after selecting to create a network game

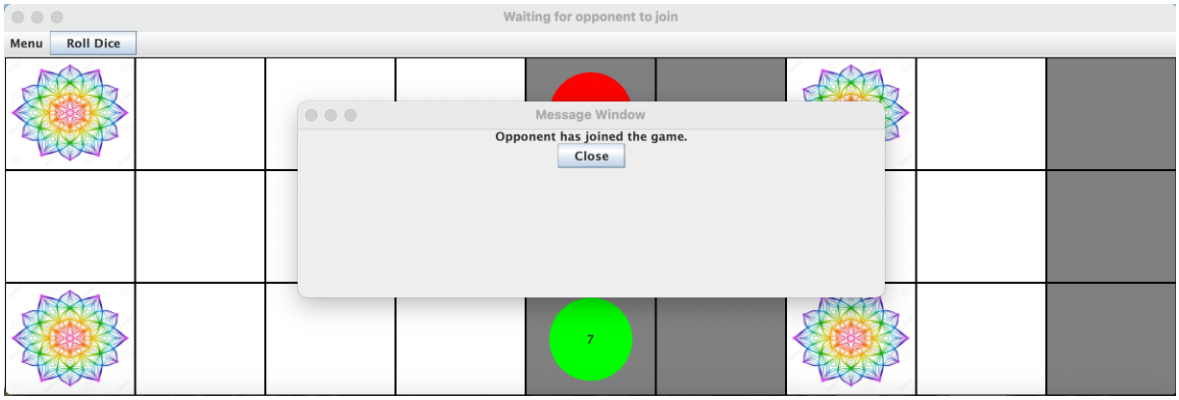

Figure 4: Showing a MessageWindow that informs the user of the fact that the other player has just joined the network game

Figure 5: Properly displaying customized MessageWindows for each player after one of them has just won the network game

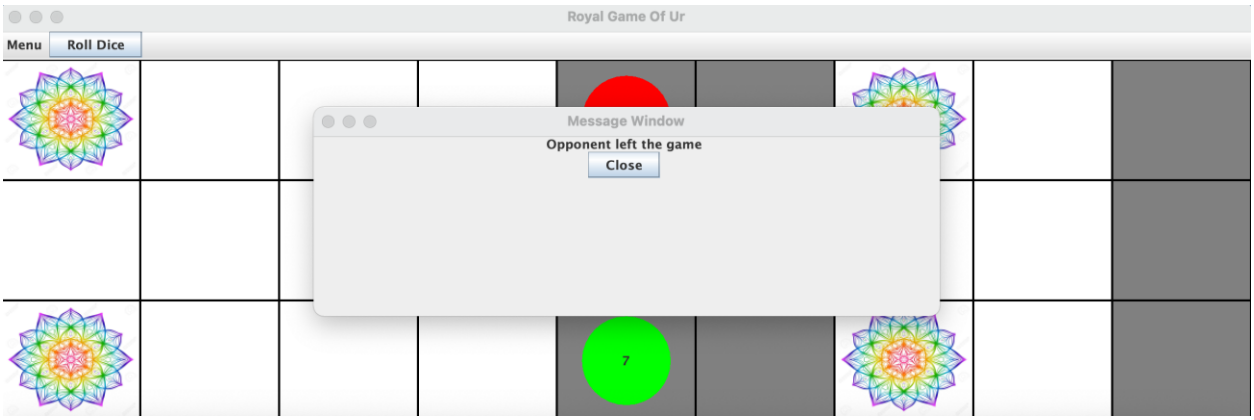

Figure 6: Properly displaying a MessageWindow to make the user aware of the fact that the opponent has just left the network game

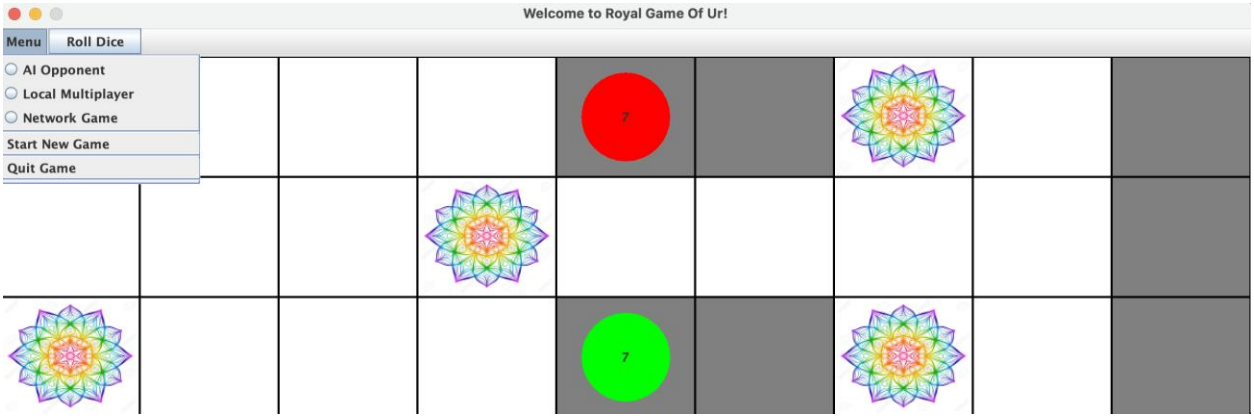

Figure 7: Showing the Game's Menu from which the user can select which game mode they want to play, or they can start a new game or quit the current one.

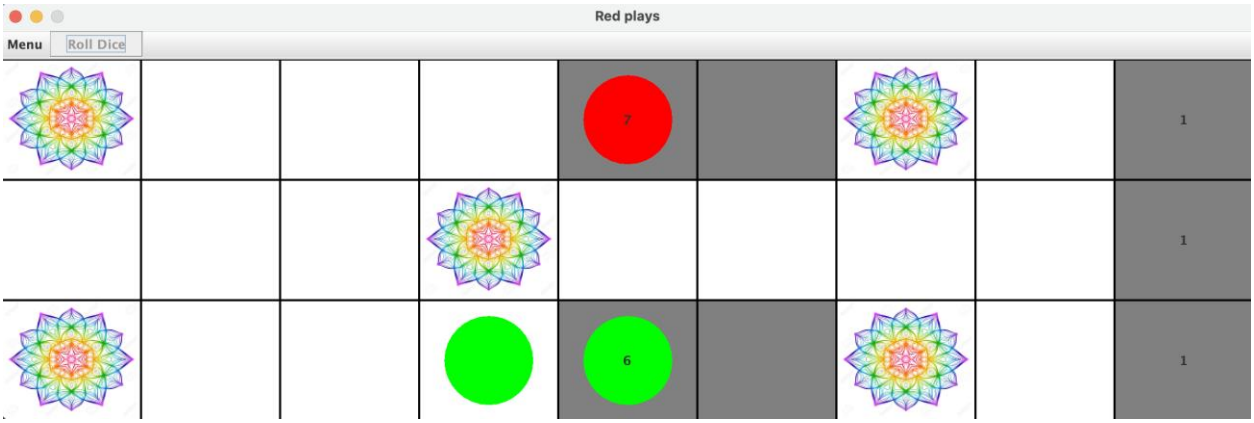

Figure 8: Showing the positioning of the dice on the board and how each die's output will be displayed after rolling them.

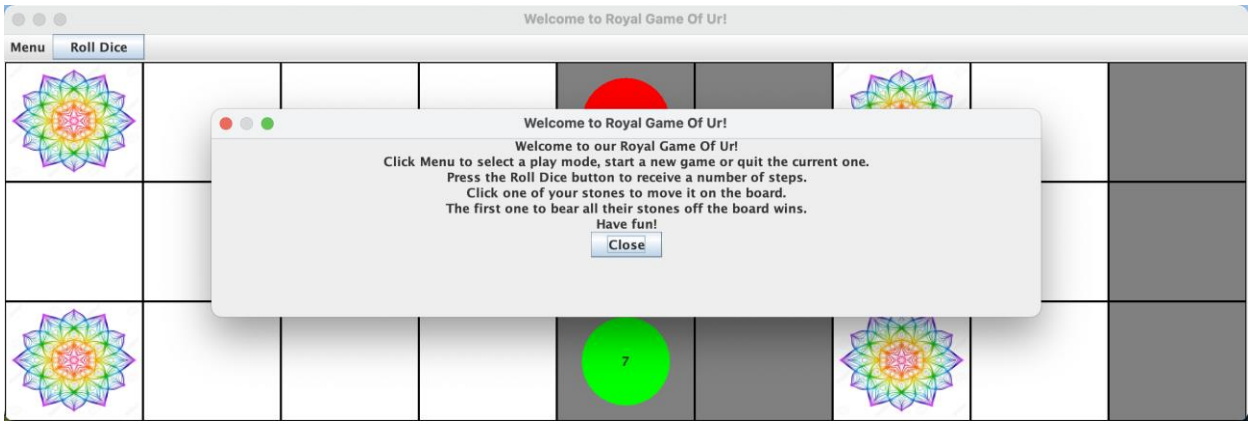

Figure 9: Showing the intro message that will be displayed every time the user runs the program

Mercurial Repository Link

https://rdn1.hg.cs.st-andrews.ac.uk/RoyalGameOfUr

Word count

Total number of words in this report: 2315 (240 of them represent the descriptions of the references and the figures above)

References

1. *resizing a ImageIcon in a JButton*. Stack Overflow. Retrieved 30 April 2021, from https://stackoverflow.com/questions/2856480/resizing-a-imageicon-in-a-jbutton

2. Retrieved 30 April 2021, from https://www.javacodex.com/More-Examples/2/14

3. *Vector Rosettes pattern. rainbow color. Vector illustration.*.. 123RF. Retrieved 30 April 2021, from https://www.123rf.com/photo_98628610_stock-vector-vector-rosettes-pattern-rainbow-color-vector-illustration-decorative-elements.html

4. Youtube.com. Retrieved 30 April 2021, from https://www.youtube.com/watch?v=ZIzoesrHHQo&t=950s

5. *javanetexamples*. Cs.lmu.edu. Retrieved 30 April 2021, from https://cs.lmu.edu/~ray/notes/javanetexamples/

6. *How do I add an image to a JButton*. Stack Overflow. Retrieved 2 May 2021, from https://stackoverflow.com/questions/4801386/how-do-i-add-an-image-to-a-jbutton