Royal Game of Ur – CS1006 Part 3

Requirements

This project required us to create a digital version of the Royal Game of Ur, featuring a graphical user interface. The game needed to prevent illegal moves and recognise when a player had won. The Royal Game of Ur’s exact rules are not known, so we were required to select one of many different rule sets to implement.

The player should be given the option to play against another person over a local network or play against an AI.

Team Communication

At our first meeting, we broke the project into three parts: networking, building the interface, and AI. This way, each person could concentrate on a specific area but still help the project move forward together.

We had regular team meetings to share updates. If someone needed help or had questions about another's work, they could have reached out to the team using a WhatsApp group.

Besides talking during meetings, we also focused on writing clear explanations and notes in our code. This helped everyone understand the code better, making it easier for us to work together.

We also set up a clear way to handle our code changes using Git, a tool for tracking changes. We decided to work on different parts at the same time using branches, which helped us avoid mixing up our work. We made it a habit to save changes often so we could see how the project evolved and work together more smoothly.

We chose to combine changes using git merge instead of git rebase to avoid mistakes and keep our project's history clear and accurate.

In short, talking often and clearly, documenting our code well, and using a smart way to handle code changes were key to our teamwork and the success of our project.

Certainly! Here's your document content without the LaTeX code:

---

Royal Game of Ur

[student ids]

April 2024

Overview

Requirements

This project required us to create a digital version of the Royal Game of Ur, featuring a graphical user interface. The game needed to prevent illegal moves and recognize when a player had won. The Royal Game of Ur’s exact rules are not known, so we were required to select one of many different rule sets to implement. The player should be given the option to play against another person over a local network or play against an AI.

Team Communication

At our first meeting, we broke the project into three parts: networking, building the interface, and AI. This way, each person could concentrate on a specific area but still help the project move forward together.

We had regular team meetings to share updates. If someone needed help or had questions about another's work, they could have reached out to the team using a WhatsApp group.

Besides talking during meetings, we also focused on writing clear explanations and notes in our code. This helped everyone understand the code better, making it easier for us to work together.

We also set up a clear way to handle our code changes using Git, a tool for tracking changes. We decided to work on different parts at the same time using branches, which helped us avoid mixing up our work. We made it a habit to save changes often so we could see how the project evolved and work together more smoothly.

We chose to combine changes using git merge instead of git rebase to avoid mistakes and keep our project's history clear and accurate.

In short, talking often and clearly, documenting our code well, and using a smart way to handle code changes were key to our teamwork and the success of our project.

Design

Our project is organized in folders, here is the description for them.

bin: Contains compiled Java classes.

build: Contains build and run scripts.

lib: External libraries used in the project.

src: Source code files in Java format.

* ai: Classes related to AI agents.
  + agent: Classes implementing various AI agents.
* board: Classes related to the game board.
* controller: Classes responsible for controlling game logic.
  + action: Classes representing game actions.
    - game: Classes representing game-related actions.
    - menu: Classes representing menu-related actions.
* exceptions: Custom exception classes.
* game: Classes representing the game and player logic.
* main: Main class to run the program.
* player: Classes related to player management.
* server: Classes related to server-client communication.
* states: Classes representing different game states.
* ui: User interface classes.

tests: Unit tests for the project.

Networking

The networking aspect of the project involved setting up communication between clients for multiplayer functionality. Our team decided to use TCP/IP-based protocol due to its reliability and ease of implementation. Eventually, we end up with the following cycle flowchart.

TODO - insert flowchart

GUI

Creating a user-friendly graphical user interface (GUI) was crucial for ensuring an enjoyable gaming experience.

AI

Testing

Evaluation

Conclusion

References

Appendix

---

You can copy this text directly into your Word document. Let me know if you need any further assistance!