Dokumentacja

1 Kompilacja oraz uruchomienie

$ *javac -d ./out/ ./src/*.java*
$ *cd out*
$ *java Main*

2 Narzędzia

- Edytor: IntelliJ IDEA, Visual Studio Code
- OS: Windows 11, Ubuntu 22.04
- System kontroli wersji GIT (GitLab)
- SDK: OpenJDK 20.0.1
- Wymagane pakiety: javax.swing, java.awt, java.util, java.io, java.nio

3 Ogólny opis projektu

Aplikacja została podzielona na 3 warstwy zgodnie ze wzorcem projektowym Model – Widok – Kontroler.

Widok (*GameView*) odpowiada za wizualizacje danych z Modelu oraz prowadzi integracje z użytkownikiem przez przyciski.

Model (*Model*) odpowiada za logikę działania aplikacji. Przechowuje dane o punktacji graczy z bieżącej tury oraz zwycięzcach. Również na bieżąco po zakończeniu rzutu każdego gracza wykonuje zapis wyników do pliku. Po zmianie gracza lub rzucie kością wymusza odpowiednie metody widoku w celu wizualizacji zmian.

Kontrolor (*GameController*) reaguje na interakcje użytkowania z widokiem i wywołuje odpowiednia metody w celu zmiany bieżącego stanu modelu, a właściwie aktualnie wybranego gracza. Na podstawie losowania wyniku każdego rzutu ustalana jest wartość oczek każdej kości oraz zapisanie tej wartości w modelu. W widoku wyświetlana jest graficzna interpretacja wylosowanej ilości oczek.

4 Obsługa programu z punktu widzenia użytkownika

W menu można wybrać ilość graczy (2, 4, 6) oraz ilość kości (1, 2, 3), a po wciśnięciu przycisku *Start* pojawia się menu kontekstowe gry. Gracz może tylko raz wylosować kości, a dopiero po wykonaniu rzutu jest możliwość zmiany gracza. Gracze są numerowani od 0. Po zakończeniu rzutu ostatniego z graczy wyłaniani są zwycięzcy. W celu rozpoczęcia nowej tury należy wcisnąć odpowiednio oznaczony przycisk.

Na bieżąco w pliku zapisywane są wyniki gry. Plik jest ”czyszczony” w momencie otwarcia aplikacji na nowo. Jeżeli nie ma pliku źródłowego wyświetlany jest komunikat w konsoli, lecz grę można nadal kontynuować bez tej funkcjonalności.

5 Wymagane elementy projektu

1. Kolekcje
   Została zastosowana kolekcja typu *ArrayList*, ponieważ zwraca ona listę zwycięzców, która może się zmieniać w zależności od wylosowanych wyników i jest dynamicznie rozszerzana. Z kolei klasyczna tablica jest zaimplementowana przy wyborze liczy graczy oraz liczby kości, gdyż wartości te się nie zmieniają.

2. Dziedziczenie
   Klasa *View* jest klasą abstrakcyjną, po której dziedziczą klasy *MenuView* oraz *GameView*. Klasa *View* implementuje *Frame'a* o określonych parametrach *final*. Służy ona do zdefiniowania paramentów okna takich jak wielkość oraz możliwość rozszerzania, z których korzystają konkretne widoki.

3. Hermetyzacja
   Klasy posiadają pola prywatne. Klasa *Model* zawiera prywatną klasę *Player*, ponieważ tylko model powinien mieć możliwość manipulacji jej danymi.

4. Wielowątkowość
   Poza GUI, które znajduję się w oddzielnym wątku, posiadamy również wątek zajmujący się zapisem wyników gry do pliku .txt. Zostało to zrealizowane przy pomocy wyrażenia lambda. Metoda *rollAndSum()* odpowiadająca za rzucanie oraz sumowanie kośćmi również znajduje się w oddzielnym wątku. Nie stosowaliśmy mechanizmów synchronizacji, ponieważ każdy wątek korzystał z oddzielnych zasobów.

5. Obsługa zdarzeń
   Przy zapisie wyników do pliku występuje obsługa zdarzeń wejścia oraz wyjścia *IOException* w przypadku nie znalezienia pliku źródłowego.

6 Schemat UML