Projekt Programowanie Komputerów						
Rok akademicki	Termin	Rodzaj studiów	Kierunek/grupa	Prowadzący	Semestr	Sekcja
2024/2025	Piątek 14:00 – 16:15	Dzienne	IPpp20	ŁM	3	1

Sprawozdanie z projektu

Data wykonania ćwiczenia: 2025-27-01

Temat ćwiczenia:

Symulator Układu Automatycznej Regulacji

Skład sekcji:

1. Michał Jaworek
2. Mateusz Rymoniak

Link do repozytorium: https://github.com/MateuszRym/pk-projekt.git

Skład sekcji i podział obowiązków

W skład sekcji projektowej wchodzą:

- Michał Jaworek
- Mateusz Rymoniak

Podział obowiązków członków sekcji:

- Michał Jaworek:
    - Projekt struktury programu, hierarchia klas;
    - Oprogramowanie funkcjonalności symulatora: ARX, generator sygnałów;
    - Zaprogramowanie działającej symulacji na podstawie zrobionych klas jej pojedynczych elementów (kompozycja Symulacja);
    - Integracja symulatora z GUI, implementacja QTimera;
    - Oprogramowanie funkcjonalności GUI;
    - Testowanie aplikacji.
- Mateusz Rymoniak:
    - Oprogramowanie funkcjonalności symulatora: PID, sprzężenie zwrotne;
    - Projekt GUI i jego realizacja;
    - Oprogramowanie funkcjonalności GUI;
    - Oprogramowanie wykresów;
    - Testowanie aplikacji.

Zmiany projektu aplikacji

- Klasy SygSkok, SygSin i SygKwad dziedziczące po SygGen, zostały zastąpione jedną klasą SygGen z trzema metodami odpowiadającymi każdemu typowi sygnału.
- Zrezygnowano z klasy Zegar.
- Zrezygnowano z globalnego licznika kroków symulacji na rzecz pola prywatnego klasy generatora sygnałów oraz pola prywatnego MainWindow do obsługi wykresów.
- Klasa symulacja, na pewnym etapie zmieniona z kompozycji na agregat, ostatecznie została kompozycją poszczególnych symulowanych obiektów UAR.

- Pozbyto się zakładki „Konfiguracja wykresów” w GUI.
- W GUI stworzono cztery osobne wykresy na różne informacje o symulacji, zamiast jednego zbiorczego.
- W GUI dodano przycisk „Resetuj” służący do przywracania nastawów symulacji do pierwotnego stanu oraz czyszczenia wykresów.

Napotkane trudności

- **Generator sygnału:** zastosowanie dziedziczenia i polimorfizmu dla klas generatora sygnału znacznie skomplikowało integrację back-endu z funkcjonalnością front-endu. W pewnym momencie, na etapie integracji kodu symulatora z GUI zastąpiono cztery klasy jedną, w której sygnały wejściowe liczone są w trzech dedykowanych różnym typom sygnałów metodach. Znacznie uprościło to implementację.
- **Wykresy:** przy pierwszej implementacji wykresów do programu popełniliśmy kluczowy błąd w tym, że najpierw dodaliśmy wykresy, a dopiero potem próbowaliśmy zaimplementować QTimer. Wykresy prawidłowo generowały dane dla pętli wykonanej na potrzeby testów, ale po dodaniu QTimera nie udało się dostosować wykresów do zmian w kodzie. Dopiero całkowite usunięcie wykresów i ponowne ich wykonanie po właściwej implementacji QTimera zapewniło poprawne działanie.
- **Regulator PID**: poprawnie działające wykresy pozwoliły nam dostrzec, że regulator PID nie działa w prawidłowy sposób: wykresy generowane dla części całkującej i różniczkującej nie pokrywały się z oczekiwaną charakterystyką. Spowodowane to było błędnym działaniem getterów, które pobierały wartości wyjściowe PID dla wykresów: gettery wywoływały metody liczące tych części regulatora, tym samym zmieniając ich pamięć i wpływając na przebieg symulacji. Aby zaradzić temu problemowi utworzono w PID prywatne pola, które przechowują wartości części I oraz D obliczone w obecnym kroku symulacji i to je pobierają gettery.

Podsumowanie - wnioski

Michał Jaworek:

Wiele błędów i problemów napotkanych w etapie produkcyjnym pomogło mi lepiej zrozumieć proces tworzenia rozbudowanej aplikacji okienkowej i istotność planowania kolejności i sposobu implementacji tych funkcjonalności. Odnośnie do organizacji pracy grupowej wnioskiem z tego projektu jest rzetelniejsza organizacja obowiązków osób pracujących nad programem – zdarzały się próby wspólnej pracy nad tym samym fragmentem kodu, co prowadziło do odmiennych podejść do rozwiązania problemu i straty czasu na zrozumienie kodu napisanego przez partnera przed kontynuacją pracy. Praca nad projektem

pomogła również lepiej docenić proste rozwiązania w projektowaniu aplikacji i zaznajomić się z funkcjonalnością środowiska Qt.

Mateusz Rymoniak:

Cały projekt przede wszystkim pokazał mi, że pisanie kodu z inną osobą tak, żeby współgrał ze sobą i działał bezbłędnie, wymaga zgoła innego podejścia niż na klasycznych zajęciach programowania czy pisaniu własnych programów; zmusza to do zwiększenia czytelności swojego kodu oraz poprawienia umiejętności analizy czyjegoś rozumowania i podejścia kogoś innego do problemu, a także współpracy przy jednym kodzie. Ważna okazała się także umiejętność czytania dokumentacji nowych klas i modułów, co przydało się podczas pracy z wykresami (QCustomPlot). Projekt ten pokazał również, że specyfika aplikacji okienkowych wymaga od programisty szerszego spojrzenia na możliwości wyboru dane użytkownikowi końcowemu; nie jest ona aż tak prostolinijna jak większość aplikacji konsolowych, które przyszło mi pisać.