Zaawansowane C++

tematy projektów

Edycja 2022/2023

Z poniższej listy wybieracie Państwo jeden temat do samodzielnej realizacji w trakcie przedmiotu. Informację o wyborze proszę zamieszczać w Moodle, w części „wstępny opis projektu”, **najpóźniej do 6 marca**, wraz z uszczegółowieniem tematu – dookreśleniem jego funkcjonalności (proszę opisać na stronę jak wyobrażacie sobie że będzie program działał, oraz zamieścić szkic interfejsu użytkownika, ew. głównych klas w programie).

W drugim etapie (**do 27 marca**) poproszę o zamieszczenie w systemie co najmniej diagramu klas programu.

Zakończone projekty zamieszczacie **na 3 dni** przed planowanym terminem obrony na moodle zamieszczając w nich:

a) kody źródłowe programu, wraz z plikiem projektu i plikami zasobów, bez wyników kompilacji, oraz dodatkowo - link do repozytorium z kodem

b) Raport z wykonania programu, zawierający aktualny diagram klas programu, oraz instrukcję obsługi programu (diagram interakcji) – o ile nie jest ona trywialny.

oraz przesyłacie mail do swojego prowadzącego z informacją o chęci przystąpienia do obrony

Wymagane jest zachowanie kolejności przesyłania zadań – nie można zamieścić etapu II jeśli nie uzyskało się oceny za etap I, i nie można przesłać etapu III jeśli nie ma oceny za etap II – dlatego też zachęcam do regularności.

Projekty muszą być napisane w języku C++, przy wykorzystaniu dowolnego środowiska czy kompilatora. Jeśli nie macie własnych preferencji – polecam bibliotekę **Qt** wraz ze środowiskiem programistycznym **QtCreator** dla programów z GUI, lub też w przypadku programów bez GUI lub z GUI tworzonym innymi technikami niż Qt – polecam **CLion** firmy **JetBrains** jako narzędzie do edycji i uruchamiania kodu (wymaga posiadania konta w domenie PW dla darmowej wersji) albo **Microsoft Visual Studio Code** – w pełni darmowy edytor. Realizacja większości projektów będzie od Was wymagała wykonania graficznego interfejsu użytkownika. Z drugiej strony – zakres materiałów które przygotowaliśmy dla Was nie pokrywa tej tematyki – tu musicie poszukać własnych źródeł.

Tematy podzielone są na grupy trudności, wg naszej subiektywnej oceny. Programy z grupy **łatwe** są przeznaczone dla osób które dotychczas nie miały dużego doświadczenia w programowaniu w C++, i starają się nadrobić zaległości. Programy z tej grupy będą oceniane na max. 40 pkt (w sumie), chyba że będą zawierały jakieś rozszerzenie podstawowej funkcjonalności (proszę je zamieszczać w przesyłanym do nas opisie).

Grupa „**średnie**” jest przeznaczona dla osób które opanowały już podstawy języka, i chcą się czegoś nowego nauczyć poprzez „rozpoznanie bojem”. Z projektu z tej grupy można otrzymać max. liczbę punktów.

Grupa „**trudne**” jest dla tych, którzy lubią spędzać długie godziny przed komputerem ;) Wykonanie opisanych tam programów wiąże się ze sporym nakładem pracy – ale też i satysfakcją po jej zakończeniu. W przypadku projektów z tej grupy zgadzamy się na pracę w **zespołach 2-osobowych** (w szczególnie skomplikowanych przypadkach 3 osoby też są dopuszczalne), pod warunkiem wykorzystania przez zespół narzędzi do wersjonowania kodu (git, przykładowo https://github.com/ lub svn, przykładowo https://riouxsvn.com/) wraz z rejestrowaniem aktywności członków zespołu. W przypadku zespołów prosiłbym też o wyraźne oznaczenie już na etapie opisu projektu, za jaką jego część odpowiadają poszczególni członkowie

W każdym przypadku będziemy wymagali, aby w Waszych projektach były następujące elementy:

- prawidłowe wykorzystanie klas – program ma być napisany przy wykorzystaniu programowania obiektowego, przy czym podział na poszczególne klasy i przypisanie im zadań mają być wykonane w sposób sensowny (w przypadku wątpliwości czy Wasze pomysły są sensowne – konsultujcie je z nami)
- w projektach musi występować dziedziczenie, w ocenionych na wyżej niż 3 – musi też być polimorfizm
- klasy mają być zapisane zgodnie z powszechnie obowiązującymi zasadami dobrego stylu – czyli – deklaracja klasy w pliku nagłówkowym (*.h), definicje metod w pliku implementacji (*.cpp), każda klasa w swoich plikach. Szablony w całości umieszczamy w nagłówkach.
- klasy mają mieć udokumentowany interfejs publiczny wg standardu doxygen. Udokumentowany interfejs publiczny oznacza zamieszczenie zarówno opisu zadań i przeznaczenia samej klasy, jak i opis jej metod publicznych i chronionych – do czego służą, jakie parametry przyjmują i co zwracają
- w każdym programie wymagane jest wykorzystanie biblioteki standardowej (STL) lub innej opartej na szablonach, nawet jeśli miałoby to być nieco sztuczne
- należy, jeżeli nie ma ku temu uzasadnionych przeciwskazań, i występuje interfejs użytkownika, rozdzielić logikę programu od interfejsu użytkownika. W zasadzie każdy program powinien być możliwy do przebudowania i uruchomienia w wersji bez interfejsu graficznego
- w celu uzyskania maksymalnej oceny – musicie przygotować i wykorzystać w programie swój własny szablon/szablony, znów – nawet jeśli miałoby to być nieco sztuczne i nie do końca optymalne.
- w celu uzyskania maksymalnej oceny – musicie też wykorzystać repozytorium kodu (typu git lub svn) od samego początku, nawet w przypadku samodzielnej pracy.

grupa „łatwe”

1. **Szachownica**
   Napisać program implementujący szachownicę i wszystkie typy figur jako obiekty, które znają swoje możliwości poruszania się. Umożliwić wykorzystanie tego programu jako wirtualnej szachownicy z kontrolą poprawności ruchów.
2. **Ślimaks**
   Jak szybko ślimaki zjadają roślinki w naszym akwarium? Ile ślimaków możemy hodować? Program symuluje zarówno rozwój roślin jak i działalność (i rozwój) ślimaków. Kto wygra!? Oczywiście nie zawsze wszystko idzie zgodnie z planem a świat opisywany jest

przez szereg parametrów (prędkość wzrostu w %, szybkość zasiedlania, żarłoczność ślimaka zależna od wieku, itd.)

3. **Toy Soldier**
   Napisać symulator oddziału naszego wojska, składający się z żołnierzy różnego stopnia reagujących na podstawowe komendy: Baczność, w lewo patrz, do szeregu, itp...
4. **Szablony**
   Napisać własną implementację szablonów bufora cyklicznego i kolejki FIFO, zabezpieczonych przed wielodostępem. Przygotować aplikację porównującą wydajność własnego rozwiązania z rozwiązaniami opartymi o STL
5. **Żaba na ulicy**
   Napisać symulator umożliwiający przejście przez ulicę żabie. Program ma umożliwiać grę dla człowieka sterującego żabą, lub samodzielne przemieszczanie się żaby wyposażonej w sztuczną inteligencję.
6. **Obiektowy model magazynu**
   W magazynie występują różne powierzchnie pozwalające na przechowywanie towarów o określonych gabarytach zgodnie z określonymi zasadami dostępu (np. kolejki FIFO, FILO, itd.). Mamy możliwość zdefiniowania magazynu z poszczególnych części składowych oraz umieszczania w nich poszczególnych towarów. Dla każdej z przestrzeni określone są pewne reguły przechowywania. W idealnym rozwiązaniu możemy wizualizować zajętość przestrzeni magazynowej.
7. **Euro 2020**
   Symulator mistrzostw Europy (lub innych, może być i świata) w piłce nożnej. Należy zamodelować drużyny piłkarskie (poszczególni piłkarze) oraz umożliwić zaplanowanie i przeprowadzenie rozgrywek. Stosujemy bardzo uproszczone zasady oraz zmniejszone składy zespołów.
8. **Kolekcjoner**
   Opracowanie bazy danych (własny silnik) służącej do przechowywania informacji o swobodnie definiowanych obiektach, które posiadamy w domu (książki, płyty CD i DVD, kolekcja figurek porcelanowych, itp.) w postaci biblioteki (np. jak SQLLite). System ma umożliwić definiowanie różnych typów przedmiotów (opisywanych przez kilka atrybutów standardowych oraz szereg swobodnie konfigurowalnych) a następnie przechowywać informacje o poszczególnych egzemplarzach. Dla poszczególnych przedmiotów zdefiniowanych może być kilka stanów, których się one znajdują (mam, pożyczone, planowany zakup, ze złomowane, itp.). Oczywiście można dokonywać różnych operacji na rzeczach oraz dokonywać wyszukiwania według określonych kryteriów.
9. **Wycieczka robotów**
   Symulacja układu mobilnych n-robotów. Roboty poruszają się w ustalonym szyku definiowanym przez użytkownika. Każdy z robotów realizuje proste reguły śledzenia pozycji względem ustalonej liczby robotów nadrzędnych (w zależności od przyjętego układu geometrycznego szyku).

grupa „średnie”

10. **Easy rider**
    Napisać program symulujący ruch uliczny na zadanej mapie. Poruszające się pojazdy muszą się wykazywać szczątkową inteligencją, z chęcią też zobaczę symulację tworzenia się korków
11. **Błędnik**
    Napisać symulator samochodów poruszających się w sposób losowy lub zadany (w sensie wyboru kolejnych punktów) po mapie ulic zgodnie z zasadami ruchu drogowego (uproszczonymi). Należy symulować losowo pieszych oraz zmianę sygnalizacji świetlnej.

12. **Układanki z lat dziecinnych**
    Napisać program symulujący "puzzle" (kwadrat o zadanej długości boków n, z polami ponumerowanymi od 1 do n*n-1, jedno pole puste. Użytkownik może przesunąć na puste pole dowolny z sąsiadujących elementów). Program powinien umożliwiać ułożenie pul wierszami od najmniejszego numeru do największego ręcznie lub automatycznie (w przypadku układania automatycznego mamy do czynienia z trudnym projektem).
13. **Micromouse**
    Napisać program umożliwiający symulację przejścia robota micromouse przez labirynt. Program ma umożliwiać wizualizację ruchu robota, sam robot ma być obiektem, ma mieć czujniki, i ma podejmować decyzje tylko i wyłącznie w oparciu o lokalne dane. Więcej informacji o micromouse znajdziecie po wpisaniu hasła w wyszukiwarkę.
14. **Doom dla opornych**
    Napisać grę podobną do starego PacMan-a w wersji umożliwiającej grę w sieci przez kilku graczy
15. **Ekstra wyświetlacze**
    Napisać zestaw kilku wyświetlaczy graficznych w bibliotece QT, możliwych do zastosowania we własnym kodzie, i implementujących typowe wyświetlacze techniczne (wskaźnik diodowy, wskaźnik poziomu, zegary, itp …)
16. **Bardzo mini SCADA**
    Przygotować aplikację umożliwiającą ciągłe wyświetlanie danych pomiarowych przesyłanych przez sieć. Program ma umożliwiać buforowanie otrzymanych danych, skalowanie wykresów, ustalanie wszystkich parametrów wyświetlania (siatka, kolory, skalowanie, rodzaj linii). Program ma umożliwiać stosowanie przetwarzania danych przez użytkownika definiowanego albo w formie pluginów (wersja A), albo w formie parsera wyrażeń matematycznych (wersja B) W skład projektu wchodzą dwa programy – serwer generujące dane i klient wyświetlający dane.
17. **Dane pomiarowe na ekranie**
    Napisać program umożliwiający umieszczanie na ekranie różnego rodzaju wyświetlaczy (zegar, pasek, liczbowy, wypełnienie kształtu, itp. ...). Wyświetlacze mają być definiowane w postaci wtyczek (plugin-ów), i mają służyć do wyświetlania danych pomiarowych przesyłanych przez sieć. Dane są generowane przez drugi program, w formie definiowanej przez użytkownika - czyli znów wtyczki …
18. **Wykresy płaskie.**
    Kreślenie wykresów znanych po angielsku contour plot lub (po polsku będzie to chyba mapa hipsometryczna). Program ma przyjmować dane w postaci macierzy, i wyświetlać ją korzystając z mapowania kolorów definiowanego przez użytkownika. Program ma umożliwiać zoom, przesuwanie wykresów, kopiowanie do schowka.
19. **Logi systemowe**
    Opracować bibliotekę umożliwiającą zapis logów generowanych w trakcie pracy programu zarówno lokalnie w pliku, jak i przesyłanie ich zdalnie poprzez sieć. Przygotować przykładowe programy korzystające z takiej biblioteki.
20. **Remote BoomBox**
    Napisać program tworzący wizualizację dla aktualnie odtwarzanego dźwięku. Wizualizacja ma odbywać się poprzez sieć (jeden komputer odtwarza dźwięk, drugi rysuje jego wizualizację). Wizualizacje różnego typu mają być obiektami.
21. **Windziarz biurowy**
    Program sterujący działaniem wind wraz z symulatorem „czasu rzeczywistego”. Zadaniem programu jest sterowanie systemem wind według określonych zasad. Istotne jest aby system obsługiwał n wind obsługujących w m grupach o pięter. Każda z grup obsługuje piętra w podanym zakresie (p_min, p_max) + parter.

22. **Windziarz DELUX**
    Program sterujący działaniem wind w systemie, w którym panele wyboru pięter umieszczone są w holach wind a nie w samych windach.
23. **Gry wojenne**
    Symulator pojedynków prowadzonych przez jednostki bojowe. System składa się z modeli kilku typów jednostek bojowych posiadającą możliwość niszczenia wroga (czynnik losowy) oraz poruszania się. Dla dostępnego zestawu jednostek istnieje możliwość zdefiniowania oddziałów dwóch walczących stron oraz przeprowadzenia symulacji potyczki.
24. **Jestem szybki jak błyskawica!**
    Symulator wyścigów Indy. Należy opracować symulatory poszczególnych samochodów (styl jazdy, parametry, zużycie, uszkodzenia) wraz z zasadami wzajemnej interakcji (wyprzedzanie) oraz model toru (parametry geometryczne). W drugiej kolejności opracować należy symulator umożliwiający przeprowadzenie wyścigu. Należy uwzględnić możliwość występowania zdarzeń losowych (np. pęknięcie opony).
25. **Wojny rdzeniowe**
    Napisać program umożliwiający interpretację prostego kodu w opracowanym języku programowania (kodu wojowników), który będzie umożliwiał „walkę" pomiędzy programami. Programy walczą na wirtualnej maszynie Touringa. Więcej informacji o arenie, kodzie, itp znajdziecie w wyszukiwarce pod hasłem Core Wars"
26. **Własna mini baza**
    Opracować kod uproszczonego, nierelacyjnego silnika bazodanowego, umożliwiającego zapis sygnałów czasowych, w których każda próbka jest zapisywana wraz z timestampem i statusem. Sygnałów może być wiele, taktowanych z różnym czasem próbkowania, natomiast wszystkie muszą być zapisywane do jednego pliku. Przygotować program porównujący wydajność Waszego rozwiązania z wybranym silnikiem baz danych, zarówno w trybie zapisu, jak i w trybie odczytu.

Grupa „trudne"

27. **Własne języki programowania.**
    Napisać program umożliwiający zestawianie na drodze graficznej i wyliczanie wartości wyrażeń arytmetycznych. Zestawianie ma się odbywać w sposób zbliżony do tego jaki możecie zaobserwować w pakiecie Simulink
28. **Security Control Studio**
    System monitorowania budynku wraz z mechanizmem zapalania świateł wykorzystujący autonomicznych agentów stwierdzających obecność intruzów. Zakładamy, że osoby upoważnione poruszają się wraz z identyfikatorami RFID. System obsługuje: czujki ruchu, kontaktrony w drzwiach, elementy sterujące oświetleniem oraz blokadą drzwi, … . Należy także opracować system do symulacji określonych scenariuszy.
29. **Węże, węże** …
    Napisać program, który umożliwi generowanie serii danych (szeregów czasowych wg algorytmu określonego przez użytkownika dla określonego przedziału i ich wyświetlanie. Algorytm ma być określany przez użytkownika w formie skryptu albo we własnym języku programowania, albo w języku Python (do wyboru jako jedna implementacja w projekcie, nie obie na raz).
30. **Komfortowe kino**
    Napisać program symulujący publiczne przestrzenie użytkowe, w tym temperaturę w pomieszczeniach, wilgotność, stężenie CO2, w zależności od liczby osób w pomieszczeniu, warunków zewnętrznych, trybu pracy urządzeń klimatyzująco/wentylujących.
31. **Ku czci Macieja, ojca Romana**
    Napisać program umożliwiający uruchomienie optymalizacji genetycznej na wielu

komputerach połączonych siecią. Program ma implementować niezależne populacje z migracją osobników między nimi.

32. **W grupie raźniej**
    Przygotować kod biblioteki umożliwiającej wykonanie optymalizacji przy wykorzystaniu algorytmu Particle Swarm Optimization, wraz z wizualizacją postępu obliczeń, oraz programem testowym korzystającym z tej biblioteki.
33. **Skryptomania**
    Opracowanie prostego języka skryptowego oraz realizacja biblioteki analizującego i wykonującego tego typu skrypty. Skrypty powinny umożliwiać obsługę: instrukcji sterujących: if, for, switch, deklarację zmiennych typu int, double, string, deklaracji funkcji o dowolnej liczbie parametrów pobieranych i zwracanych.
34. **Mission Impossible**
    Gra, w której uciekamy przed grupą autonomicznych agentów losowo (ale nie głupio) przeczesujących teren. Po nawiązaniu kontaktu agent zaczyna podążać naszym śladem. Mamy szansę go zgubić ponieważ jesteśmy szybsi, ale ... Centrala dowodzenia czuwa, koordynuje pracę agentów zlecając im zadania nadrzędne i pobierając od nich informacje o naszym położeniu (jak nas zauważą).
35. **Matrix**
    Opracować symulator robota człekokształtnego uchylającego się przed lecącymi na niego pociskami. Poszczególne człony robota symulowane są jako niezależne jednostki ale sterowane centralnie (oczywiście przy spełnieniu pewnych ograniczeń).
36. **SI**
    Celem programu jest identyfikacja lub klasyfikacja szeregów czasowych w oparciu o sztuczne sieci neuronowe tworzone przy wykorzystaniu biblioteki Google TensorFlow. Program na umożliwiać nauczenie sieci modelu, oraz jego późniejszą symulację.