Sprawozdanie z projektu

Gra Memory (C++/wxWidgets)

Seweryn Ładniak, Maciej Szrek

6 lipca 2025

Projekt zaliczeniowy – Programowanie Obiektowe (Code::Blocks + wxWidgets)

Podział zadań w zespole

Projekt realizowany był zespołowo:

- **Seweryn Ładniak**: implementacja logiki gry, testowanie, przygotowanie screenów, kompilacja programu, spakowanie EXE przez UPX.
- **Maciej Szrek**: rozbudowa dokumentacji, przygotowanie szczegółowego opisu projektu, kontrola jakości oddawanych plików, weryfikacja zgodności z wymaganiami prowadzącego.

Spis treści

1 Cel i założenia projektu

Celem projektu było samodzielne zaprojektowanie oraz implementacja klasycznej gry **Memory** (gra w pary), w języku C++ z użyciem biblioteki **wxWidgets**. Projekt miał sprawdzić praktyczne umiejętności programowania obiektowego, pracy z GUI oraz organizacji zespołowej. Ważnym celem była również nauka profesjonalnego przygotowania projektu pod kątem wymagań dydaktycznych – przenośność, zgodność z wytycznymi prowadzącego oraz przejrzysta dokumentacja.

Podstawowe wymagania funkcjonalne:

- Losowe rozmieszczenie kart na planszy (każda para w innym miejscu).
- Obsługa kliknięć – odkrywanie kart, weryfikacja par.
- Pary odkryte pozostają widoczne, nietrafione są zakrywane po krótkim czasie.
- Zliczanie ruchów oraz pomiar czasu gry.
- Okno z podsumowaniem po znalezieniu wszystkich par.
- Przejrzysty interfejs graficzny, czytelne komunikaty.

2 Opis działania gry

Gra Memory polega na odsłanianiu par zakrytych kart. Gracz odkrywa na planszy po dwie karty; jeśli są one identyczne – pozostają odkryte. Nietrafione karty po krótkim czasie są zakrywane z powrotem. Gra kończy się, gdy wszystkie pary zostaną odkryte, a program wyświetla podsumowanie liczby ruchów oraz czas rozgrywki.

Ważniejsze aspekty logiki:

- Program blokuje odkrywanie kolejnych kart do czasu rozstrzygnięcia obecnej tury (dwie karty naraz).
- Odtwarzana jest logika tury – kolejne kliknięcie jest możliwe dopiero po rozpatrzeniu ostatniego ruchu (błędna para: odwrócenie po czasie).
- Aplikacja nie pozwala odkrywać tej samej karty dwa razy w tej samej turze.
- W momencie odkrycia ostatniej pary pojawia się komunikat podsumowujący: liczba ruchów i łączny czas.

3 Architektura programu i podział na klasy

Projekt został podzielony na kilka kluczowych klas:

1. Klasa `memorygraApp`

Odpowiada za cykl życia aplikacji, start programu oraz inicjalizację głównego okna. **Metody:**

- `OnInit()` – inicjalizuje aplikację, otwiera główne okno gry.

2. Klasa `memorygraMain`

Jest głównym oknem gry. Odpowiada za:

- Tworzenie planszy z kartami (przyciski ‘wxButton‘ w siatce).
- Losowanie rozmieszczenia kart.
- Obsługę kliknięć na karty.
- Zliczanie ruchów i czasu.
- Obsługę końca gry i wyświetlanie podsumowania.

Wybrane atrybuty:

- `std::vector<wxButton*> cards;` – kontener ze wszystkimi przyciskami (kartami).
- `wxButton* firstCard;` – wskaźnik na pierwszą odsłoniętą kartę w turze.
- `wxButton* secondCard;` – wskaźnik na drugą kartę w turze.
- `wxTimer* flipBackTimer;` – timer do odwracania błędnych par.
- `int moveCount;` – liczba wykonanych ruchów.
- `time_t startTime;` – czas rozpoczęcia gry.

Główne metody:

- `void OnCardClick(wxCommandEvent& event);` – obsługa kliknięcia na kartę.
- `void CheckForMatch();` – sprawdza, czy dwie odsłonięte karty stanowią parę.
- `void FlipBack();` – odwraca nietrafione karty po upływie czasu.
- `bool AllCardsMatched();` – sprawdza, czy wszystkie karty są już odkryte.
- `void ShowWinDialog();` – wyświetla okno podsumowania po zakończeniu gry.

3. Zarządzanie zasobami i konfiguracją

Projekt korzysta z pliku zasobów `resource.rc`, który dołącza ikonę programu oraz plik `wx/msw/wx.rc` dla prawidłowego działania wxWidgets na Windows. Ikona jest ustawiona za pomocą wpisu w resource.rc:

```
IDI_ICON1 ICON "memory.ico"
#include "wx/msw/wx.rc"
```

Wszystkie ścieżki do wxWidgets w pliku projektu są podawane przez zmienną globalną `$(#wx)`, co zapewnia przenośność projektu.

4 Tabela testów i scenariuszy

5 Opis interfejsu użytkownika

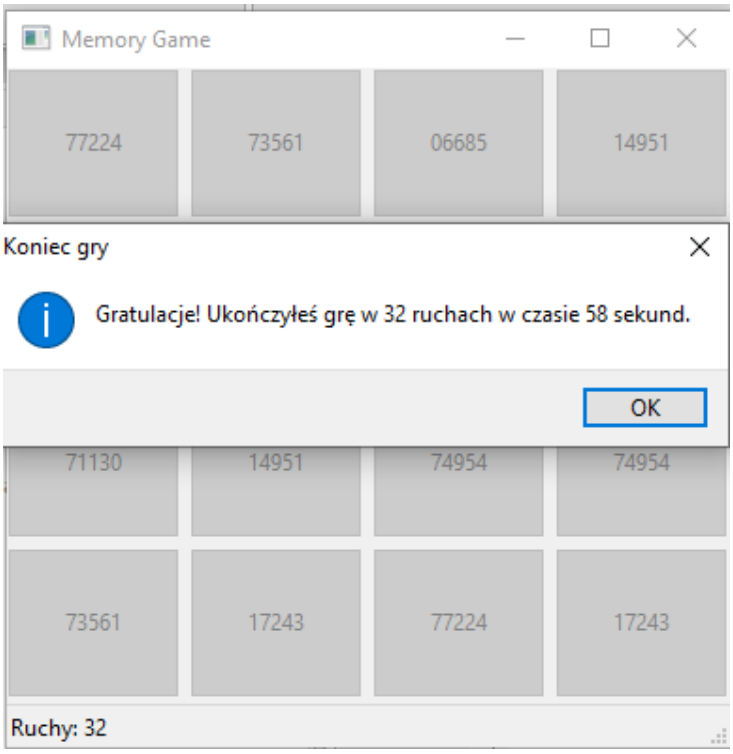
Aplikacja posiada intuicyjny interfejs graficzny zbudowany na bazie komponentów wxWidgets:

- plansza gry z siatką przycisków-kart,
- licznik ruchów i licznik czasu,
- komunikaty o błędnych ruchach i zwycięstwie,
- możliwość zamknięcia gry w dowolnym momencie.

Test	Oczekiwany rezultat	Wynik
Poprawne odkrywanie par	Odsłonięte identyczne karty zostają odkryte	OK
Błędne pary	Nietrafione karty zakrywają się po sekundzie	OK
Kliknięcia szybkie i powtórzone	Nie można odsłonić więcej niż 2 kart na raz	OK
Podwójny klik tej samej karty	Blokada ponownego wyboru tej samej karty w turze	OK
Zakończenie gry	Pojawia się podsumowanie z czasem i ruchem	OK

Tabela 1: Tabela wybranych testów gry Memory

Ekran startowy



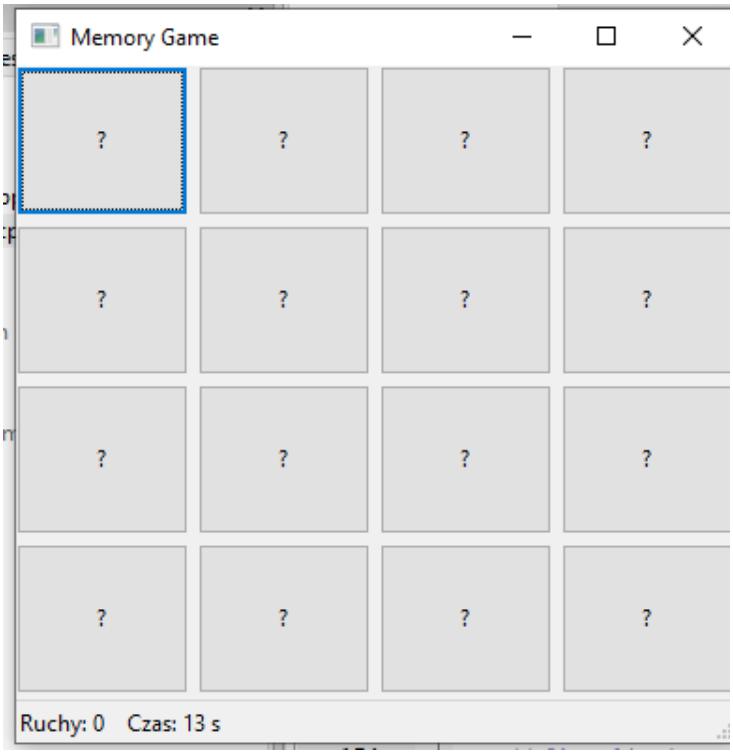
Rysunek 1: Okno startowe gry Memory (wszystkie karty zakryte).

Komunikat zwycięstwa

6 Instrukcja budowania i uruchamiania programu

- System operacyjny: Windows 10/11 (zalecane)
- Code::Blocks z pluginem wxSmith
- Biblioteka wxWidgets 3.2.8 (skompilowana na lokalnym komputerze)
- Program UPX do pakowania plików wykonywalnych

Kroki budowania:



Rysunek 2: Podsumowanie po zakończeniu gry – liczba ruchów i czas.

1. Skonfiguruj zmienną globalną `wx` w Code::Blocks – ustaw ją na katalog wxWidgets 3.2.8.
2. Otwórz plik `memorygra.cbp` w Code::Blocks.
3. Sprawdź ustawienia ścieżek w projekcie – muszą być wpisy typu `$(#wx)\include` .
4. W menu Build wybierz **Release**, kliknij **Rebuild**.
5. Po poprawnej kompilacji plik `memorygra.exe` będzie w `bin/Release/`.
6. Skopiuj plik `memorygra.exe` do folderu z UPX i wpisz w terminalu: `upx -best memorygra.exe`
7. Przetestuj EXE na komputerze bez wxWidgets.

7 Lessons learned i wnioski końcowe

Projekt nauczył nas nie tylko programowania obiektowego i pracy z bibliotekami GUI, ale także dobrych praktyk inżynierskich: - znaczenia czystości projektu i przenośności (zmienne globalne, brak śmieci w ZIPie), - profesjonalnej komunikacji w zespole, - umiejętności przygotowania projektu pod rygorystyczne wymagania formalne (czytelność, wersjonowanie, czyszczenie projektu).

8 Podsumowanie i refleksje zespołu

Projekt gry Memory w środowisku C++/wxWidgets okazał się nie tylko wyzwaniem programistycznym, ale również ciekawym doświadczeniem pracy zespołowej, inżynierskiej i organizacyjnej. Poniżej zamieszczamy szerokie podsumowanie naszych działań, wniosków oraz inspiracji na przyszłość.

Wkład pracy własnej

Projekt wymagał od nas samodzielnego przemyślenia całej architektury, podziału obowiązków, bieżącej komunikacji oraz szybkiego reagowania na zmieniające się wytyczne prowadzącego. Od pierwszych ustaleń dotyczących wyboru technologii, przez uzgadnianie podziału ról, aż po wielokrotne testy i poprawki – każda faza realizacji projektu rozwijała nasze umiejętności techniczne i interpersonalne.

W toku prac opanowaliśmy m.in.:

- tworzenie aplikacji desktopowej z własnym GUI na bazie bibliotek wxWidgets,
- efektywne wykorzystanie wzorców programowania obiektowego,
- zarządzanie zasobami projektu (ścieżki, pliki, ikony, zasoby `rc`),
- organizację kodu pod kątem łatwości rozbudowy i testowania,
- rozwiązywanie problemów kompilacyjnych i konfiguracyjnych w Code::Blocks,
- przygotowanie kodu pod statyczne linkowanie oraz pakowanie UPX,
- dokumentowanie pracy własnej z myślą o czytelniku, prowadzącym i innych programistach.

Współpraca i komunikacja w zespole

Szczególnie istotna była dla nas kwestia podziału ról i komunikacji:

- **Seweryn** był odpowiedzialny głównie za implementację logiki gry, testy, przygotowanie zrzutów ekranu oraz końcową kompilację projektu,
- **Maciej** przejął zadania związane z rozbudową dokumentacji, analizą wymagań oraz końcową kontrolą jakości i spójności projektu.

Wspólnie podejmowaliśmy decyzje o implementacji, sposobie testowania i finalnej postaci dokumentacji. Wielokrotnie wymienialiśmy się plikami (np. przez maila lub repozytorium), konsultując nawet drobne szczegóły, by efekt końcowy spełniał oczekiwania dydaktyczne i praktyczne.

Problemy i ich rozwiązywanie

Projekt nie był pozbawiony trudności – na różnych etapach pojawiły się m.in.:

- Problemy z konfiguracją środowiska i ścieżkami do bibliotek,
- Kłopoty z poprawnym statycznym linkowaniem wxWidgets,
- Nieoczywiste błędy związane z kodowaniem plików (UTF-8 vs. Windows-1250),
- Konieczność wielokrotnego czyszczenia projektu (usuwanie śmieci, plików binarnych, backupów),
- Synchronizacja plików i wersji, by uniknąć konfliktów przy oddawaniu,
- Zgodność plików graficznych (screenów) z formatem wymaganym przez LaTeX.

Każde z tych wyzwań nauczyło nas czegoś nowego – od czysto inżynierskich „myków”, przez praktyczne podejście do wersjonowania i porządkowania projektu, po lepszą komunikację w zespole.

Nauka i inspiracje na przyszłość

W trakcie projektu nauczyliśmy się:

- Jak projektować prostą, ale użyteczną architekturę klasycznej gry z zachowaniem czytelności i modularności kodu,
- Jak zarządzać większym projektem C++ w Code::Blocks i dbać o przenośność,
- Jak przygotować pełną, zrozumiałą dokumentację techniczną z przykładami kodu, screenami, opisami i instrukcją obsługi,
- Jak rozwiązywać realne, nieopisane w podręcznikach problemy – np. z dziwnymi znakami, niekompatybilnością pluginów, kodowaniem czy linkowaniem,
- Jak pracować zespołowo w trybie „zdalnym", wymieniając się kodem, testami i poprawkami,
- Jak skutecznie czyścić i finalizować projekt do oddania na uczelnię.

Dodatkowo zdobyte umiejętności i wypracowane praktyki z pewnością przełożą się na nasze przyszłe projekty – zarówno na studiach, jak i w pracy zawodowej.

Propozycje rozwoju gry i projektu

Choć oddajemy grę w wersji bazowej, widzimy szerokie możliwości jej dalszego rozwoju:

- dodanie trybu dla wielu graczy (z rankingiem i zapisem wyników),
- rozbudowanie efektów graficznych oraz animacji przewracania kart,
- wprowadzenie poziomów trudności, wyboru rozmiaru planszy lub liczby par,
- przygotowanie wersji sieciowej z możliwością gry online,
- dodanie opcji „undo", automatycznego zapisywania stanu gry,
- integracja z bazą danych wyników lub statystyk.

Podsumowanie końcowe

Podsumowując – projekt był bardzo wartościowym doświadczeniem, które pokazało nam, jak wiele różnych kompetencji można zdobyć i rozwinąć podczas pracy nad (pozornie prostą) aplikacją zespołową. Doceniamy wartość dobrze napisanej dokumentacji, czystości repozytorium oraz sensownego podziału obowiązków. Wierzymy, że nasza praca spełnia wymogi dydaktyczne i będzie czytelna oraz przydatna także dla innych studentów w przyszłości.

Seweryn Ładniak **Maciej Szrek**

Największym wyzwaniem okazało się przygotowanie środowiska do statycznej kompilacji oraz synchronizacja działań zespołu przy końcowym etapie oddawania projektu.

W przyszłości warto rozważyć:

- Implementację trybu dla wielu graczy z zapisem wyników.
- Dodatkowe efekty graficzne lub animacje.
- Możliwość wyboru poziomu trudności (więcej par, większa plansza).
- Implementację wersji sieciowej z rankingiem online.

Załączniki

- Kod źródłowy (`.cpp`, `.h`, `.cbp`)
- Plik zasobów `resource.rc`, ikona `memory.ico`
- Dokumentacja (`sprawozdanie.pdf`), screeny (`screens/`)

Podpisy

Seweryn Ładniak **Maciej Szrek**