Bartosz Błażewicz, Kamil Czapla, Józef Kasprzycki	Inżyniera Obliczeniowa	Rok 1, grupa 1.
15.01.2026 rok	Projekt – Inżynieria Obliczeniowa – grupa 1	

Nazwa projektu

Kilka dni po podziale na grupy powstały na Messengerze i na Discordzie robocze grupy o nazwie *Projekt: Gra*. Wybór nazwy został odłożony na bliżej nieokreślony czas w przyszłości, w okolicy oddawania projektu. Taki stan rzeczy utrzymuje się do dnia dzisiejszego, więc oficjalną nazwą naszej gry pozostaje *Projekt. Gra*.

Cel i opis projektu

W czasie tworzenia naszej gry inspirowaliśmy się grą o nazwie *The Binding of Isaac* . Gra realizuje podstawowe założenia typu gry o nazwie PACMAN albo Maze Runner (jak zwał, tak zwał).

Zastosowane techniki programistyczne

Napisana przez nas gra wykorzystuje w znacznym stopniu programowanie obiektowe, realizowane za pomocą klas i obiektów. Wykorzystany przez nas schemat dziedziczenia klas został zamieszczony na końcu sprawozdania ze względu na ilość wykorzystanych klas. Większość klas przez nas wykorzystanych to klasy pochodne (64% wszystkich klas) – nie liczę kilku struktur danych niebędących klasami. Każda z klas składa się z prywatnych atrybutów (tylko niektóre są oznaczone klauzulą *protected*) i publicznych metod. Dzięki użyciu enkapsulacji (hermetyzacji) danych mamy pewność, że gra nie wywróci się przez przypadkową zmianę wartości atrybutów. Wykorzystanie klas i idei programowania obiektowego nie powinno dziwić, zwłaszcza że wykorzystana została biblioteka obiektowa SFML. Dodatkowo do zarządzania projektem użyliśmy dostępnego na licencji BSD oprogramowanie Cmake[1].

Dodatkowo wykorzystaliśmy szerokie możliwości pracy na plikach tekstowych (głównie plików manifestu z rozszerzeniem .json).

Wykorzystane biblioteki - SFML

W projekcie wykorzystaliśmy graficzną bibliotekę SFML, przeznaczoną do pracy w języku C++. Nie jest ona dostarczana w standardowej wersji języka. Aby móc z niej korzystać, należy ją zainstalować w systemie. Sposób jej instalacji zależy od posiadanego systemu.

Linux (dystrybucje debianowe – Debian, Ubuntu, Mint)

Aby zainstalować bibliotekę, należy zalogować się do systemu kontem z uprawnieniami administratora, a następnie w terminalu wpisać polecenie:

```
sudo apt install libsfml-dev
```

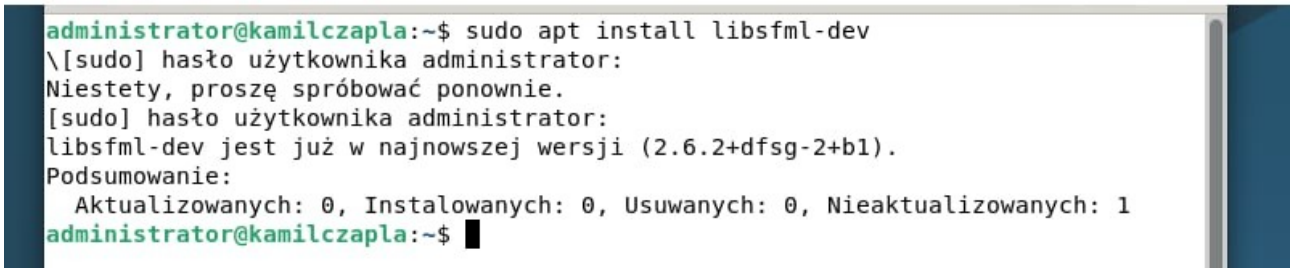
lub w starszych wersjach:

1 Oprogramowanie Cmake instaluje się tak, jak każdą aplikację, niezależnie od systemu operacyjnego. W systemie Windows należy jeszcze Cmake`a dodać do zmiennych środowiskowych konta (lub systemu), aby móc z niego korzystać.

```
sudo apt-get install libsfml-dev
```

i potwierdzić klawiszem Enter. Pojawi się prośba o wpisanie hasła użytkownika, a następnie lista instalowanych zależności. Instalację potwierdzamy wpisaniem znaku T.

Najczęściej biblioteka jest instalowana w czasie instalowania kompilatora g++, narzędzie *cmake* lub zbioru pakietów *build essential*. Jeżeli system już ma zainstalowaną bibliotekę, wywołanie powyższego polecenia zwróci informację, że SFML jest już zainstalowany:



Następnie tworzymy folder z projektem i plikiem .cpp. W tym folderze umieszczamy plik *CmakeLists.txt*, który dla samego SFML-a powinien zawierać następującą treść:

```
find_package(SFML 2.6 REQUIRED
    system
    window
    graphics
    audio
    network
) # SFML v. 2.6 (nie 3.0!)
```

Windows (10/11)

Aby móc cokolwiek zrobić, należy najpierw pobrać odpowiednią wersję bibliotekę SFML (to jest albo dla programu Visual Studio, albo dla kompilatora MinGW) ze strony internetowej dewelopera: https://www.sfml-dev.org/.

Następnie plik wypakować w wybranym przez siebie miejscu (najlepiej tak, aby można było się do niego łatwo dostać – odradzamy folder *Pobrane*). Następnie przechodzimy do konfiguracji.

1. Visual Studio 20...

Tworzymy nowy projekt (albo C++, albo Cmake`a). Następnie przechodzimy do jego właściwości – prawym przyciskiem myszy klikamy na projekt w eksploratorze rozwiązań i wybieramy pozycję *Właściwości*. Upewniamy się, że w oknie, które się wyświetli, mamy ustawione *Wszystkie konfiguracje* i *Wszystkie platformy* (w górnej części okienka). Następnie w lewej części okna wybieramy *Katalogi VC++* i dodajemy ścieżki do SFML-a w miejscach *Katalogi odwołań* i *Katalogi bibliotek* (w kolejności ścieżka do folderu *include*, ścieżka do folderu *lib*). Następnie w zakładce *C/C++ - Ogólne* na pozycji *Dodatkowe pliki nagłówkowe* wklejamy ścieżkę do folderu *include* z katalogu SFML. W zakładce *Konsolidator – Ogólne* na pozycji *Dodatkowe katalogi bibliotek* wklejamy ścieżkę do folderu *lib* naszej biblioteki graficznej.

Teraz musimy w naszym edytorze dodać zależności, aby SFML mógł działać. Zmieniamy aktualną konfigurację na *Release*, a następnie w zakładce *Konsolidator* na pozycji *Dodatkowe zależności* do-

klejamy do już obecnych nazwy plików z rozszerzeniem .lib: sfml-graphics.lib; sfml-window.lib; sfml-audio.lib; sfml-network.lib; sfml-system.lib;. Następnie zmieniamy aktualną konfigurację na *Debug* i w zakładce *Konsolidator* na pozycji *Dodatkowe zależności* doklejamy do już obecnych nazwy plików z rozszerzeniem .dll: sfml-graphics-d.lib; sfml-window-d.lib; sfml-audio-d.lib; sfml-network-d.lib; sfml-system-d.lib;.

Ostatnią rzeczą, którą należy zrobić, to skopiować pliki z rozszerzeniem .dll z folderu *bin* biblioteki graficznej do folderu z naszym projektem, za wyjątkiem pliku o nazwie *openal32.dll*. W tym momencie wszystko jest gotowe do pracy.

Aby konfiguracja przebiegła sprawnie, polecamy tutorial wideo (materiał audiowizualny jest przyjemniejszy niż tekst) dostępny pod adresem: https://www.youtube.com/watch?v=4k9ba8PIvKU

2. VS Code

Aby skonfigurować bibliotekę dla Visual Studio Code, musimy posiadać jakieś narzędzie do kompilacji (np. gcc lub MinGW) dodane do zmiennych środowiskowych. Tutaj zajmę się przypadkiem konfiguracji przez narzędzie Cmake.

Tworzymy nowy folder projektu i plik z rozszerzeniem .cpp. W folderze tworzymy plik o nazwie *CmakeLists.txt*, który powinien zawierać następującą treść (to tylko dla SFML, pozostała część pliku dotyczy już samego oprogramowania Cmake):

```
find_package(SFML 2.6 REQUIRED
    system
    window
    graphics
    audio
    network
) # SFML v. 2.6 (nie 3.0!)
```

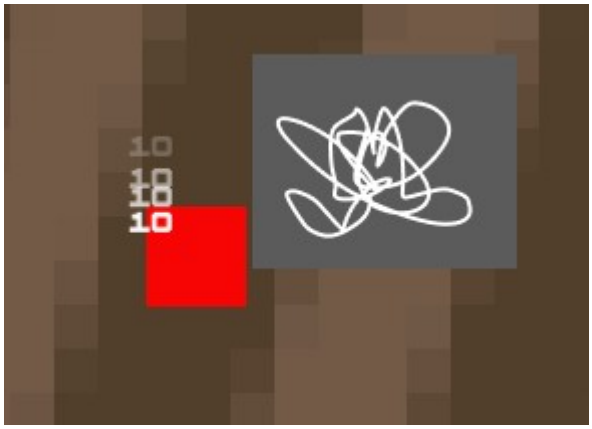
Wybrane funkcjonalności

1. Ekran logowania/rejestracji

```
case AuthState::LoginInputUser:
    // Walidacja natychmiastowa: czy taki użytkownik w ogóle istnieje?
    if (userManager.userExists(currentInput)) {
        tempUsername = currentInput;
        currentState = AuthState::LoginInputPass;
        infoText.setString("LOGIN: Enter Password:");
        currentInput.clear();
    }
    else {
        errorText.setString("User does not exist!");
        // Zostajemy w tym samym stanie, użytkownik może poprawić login
    }
    break;

case AuthState::LoginInputPass:
    if (userManager.login(tempUsername, currentInput)) {
        // Logowanie udane - sprawdź czy to Admin
        GameScreen::setAdminMode(tempUsername == "Admin");
        finished = true;
    }
    else {
        errorText.setString("Invalid password! Press Enter to restart.");
        currentState = AuthState::Menu;
        infoText.setString("1. Create New Character\n2. Login\n3. Exit");
    }
    currentInput.clear();
    break;
```

2. Wyświetlanie powiadomień na ekranie – na początku czerwony tekst informujący o rozpoczęciu poziomu, w trakcie gry uciekające punkty HP nad obiektem (czerwone nad graczem, białe nad przeciwnikiem



3. Aktualne statystyki – wyświetlane w lewym górnym rogu okna gry

4. Tryb administratora – brak obrażeń gracza.

```
void PlayerBase::takeDamage(int dmg) {
    // Jeśli GodMode (Admin) lub tymczasowa nietykalność -> brak obrażeń
    if (godMode) return;
    if (invincibilityTimer > 0.f) return;
```

Jak widać, mimo że gracz został najechany przez przeciwnika, to nie zmniejszają się punkty HP.

5. Szczegóły techniczne – kompatybilność z Linuxem i Windowsem (wykorzystanie dyrektywy #if defined (SYMBOL) do zapewnienia właściwej lokalizacji przechowywanie danych gry)

```
#if defined(__linux__)
    : userManager("../data/users.json"), // Na linuxie wychodzi tylko jeden folder wyżej
#else
    : userManager("../../data/users.json"), // Na windowsie dwa foldery
#endif
```

W powyższym fragmencie kodu zapewniamy, że jeżeli zdefiniowanym symbolem jest __linux__, to gra będzie szukała pliku z danymi użytkownika, wychodząc jeden folder wyżej i przechodząc do odpowiedniego podfolderu. W przypadku braku tego symbolu aplikacja będzie szukała odpowiednich plików, wychodząc dwa foldery wyżej i przechodząc dalej do odpowiedniego podfolderu.

Link do repozytorium

https://github.com/jozef-kasprzycki/agh-sfml-game

Instrukcja uruchomienia

Sposób I

Pobieramy repozytorium gry z adresu https://github.com/jozef-kasprzycki/agh-sfml-game (może być w pliku .zip), skanujemy na obecność wirusów lub innych zagrożeń[2] i rozpakowujemy na komputerze. W folderze głównym gry tworzymy nowy folder o nazwie *build* i przechodzimy do niego. Otwieramy konsolę i w wierszu poleceń wpisujemy najpierw polecenie `cmake ..`, a następnie polecenie `make`. Po pomyślnym procesie kompilacji uruchamiamy plik wykonywalny (na Linuxie z rozszerzeniem .out, na Windows z rozszerzeniem .exe). Sterowanie postacią na ekranie odbywa się za pomocą klawiszy WSAD lub strzałek na klawiaturze.

Wadą tego sposobu jest konieczność posiadania na komputerze zainstalowanej (lub pobranej i rozpakowanej w wiadomym miejscu) biblioteki SFML oraz zainstalowanego Cmake`a. Dla osób, które chcą tylko zagrać w naszego MazeRunnera, polecamy drugi sposób uruchomienia gry.

2 Korzystanie z internetu wiąże się z ryzykiem zwiększonej podatności na liczne zagrożenia obecne w sieci. Nie sugerujemy, że przygotowany przez nas projekt został zainfekowany wirusami, ale nie bierzemy odpowiedzialności za ewentualne szkody powstałe w wyniku zainfekowania wirusami.

Sposób II

Aby ułatwić uruchomienie projektu na komputerze, został przygotowany instalator gry na system Linux (dystrybucje debianowe). Wystarczy pobrać instalator, przeskanować na obecność wirusów lub innych zagrożeń i uruchomić. Następnie postępujemy zgodnie z instrukcją wyświetlaną na ekranie komputera.

Załącznik 1 – schemat dziedziczenia klas (nieaktualny!!!)