Imię i Nazwisko Lena Janik Karolina Drost Bartosz Dumański	**Kierunek** IO	**Rok studiów i grupa** **1, 1**
Data zajęć 16.10.2025	**Numer i temat sprawozdania** „Traktorzysta” – projekt gry	

1. „Traktorzysta” – cel i opis gry

- **Fabuła**:

Po zeszłorocznych dożynkach na dotychczas spokojnych wsiach wybucha konflikt. Sołtys Suchej Psiny postanawia sabotować udział wsi Zimna Wódka w konkursie na najlepszą koronę dożynkową. W tym celu zwołuje Koło Gospodyń Wiejskich (KGW) i razem wyruszają na pola w Zimnej Wódce, aby przeszkodzić w zbiorach i przygotowywaniu korony.

- **Cel gry:**

Gracz wciela się w role rolnika z Zimnej Wódki, przechodzi poziomy zbierając plony i korony, jednocześnie uciekając przed zawistnymi babami i sołtysem Suchej Psiny.

2. Zastosowane techniki programistyczne

1) Programowanie Obiektowe

- Gra posiada wiele struktur, takich jak m. in. Player, Enemy czy Boss. Każda z nich posiada swoje własne pola (np. prędkość bądź rozmiar)
- Kluczową role w naszej grze pełni struktura SDL_Rect, którą możemy opisać jako „szkielet” każdego obiektu. Pomaga w opisywaniu obiektów oraz ich geometrii.

2) Algorytmy i struktury danych

- Pathfinding (BHS – Breadth First Search) – jest to algorytm zalewowy, służący w grze do wyznaczenia przeciwnikom optymalnej trasy do gracza. Działa w oparciu o kolejke, gdzie dodajemy współrzędne pola, na którym stoi wróg oraz o Mape rodziców, gdzie dla każdego pola zapisywana jest informacja z którego pola do niego przyszliśmy.

- Wektory – czyli dynamiczne zarządzanie parametrami klas

- Maszyna stanów – dynamiczne przełączanie między stanami gry (menuAcitve, levelCompleteScreen, gameOver)

3) Grafika

- Przezroczystość – w celu osiągnięcia przeźroczystości korzystamy z funkcji wczytajTeksture, która za pomocą podmienienia kolor Magenty zapewnie naszym grafikom przeźroczystość.

- Paralaksa – w oknie podziękowań zastosowaliśmy efekt paralaksy, powszechnie znany ze scen kończących film

4) Mechanika Gry

- Pętla Gry – skorzystanie z pętli while(running) oraz skorzystanie z synchronizacji SDL_Delay(16), co zapewnia stałe 60 fpsów.

- System kolizji – W naszej grze zastosowaliśmy funkcje biblioteki SDL_HasIntersection, wykrywującą kolizje między graczem a punktami, przeciwnikami a przeszkodami

- Budowa Labiryntu – wykorzystaliśmy tablice dwu wymiarową, służącą do przechowywania wzoru labiryntu. Komórka zawierająca cyfre 0, oznacza pole po którym gracz może chodzić oraz zbierać zboże. Natomiast cyfra 1 oznacza pole, gdzie występuje ściana.

- System boosterów - gracz ma możliwość zebrania ulepszeń, które mają na celu urozmaicenie rozgrywki

- Walka z bossem – na 3 poziomie gracz napotyka bossa. Jego zadaniem jest zebranie wszystkich koron, jednocześnie unikając kontkatu z sołtysem (naszym bossem)

- System osiągnięcia – po wygraniu potyczki z bossem, gracz otrzymuje osiągnięcie za pokonanie bossem, co jest równoczesne z przejściem gry. Osiągnięcie zostaje pernamentnie umieszczone w menu głównym.

- System personalizacji traktora

5) Obsługa zasobów i systemów zewnętrznych

- Zapisywanie danych – zapis oraz odczyt danych zapisywany w plikach tekstowych, przykładem może służyc plik postep.txt, który służy do zapisywnia postępów gracza

- Sterowanie – obsługa klawiatury oraz myszki

- Randomizacja – użycie funkcji Shuffle() do losowania rozmieszczenia boosterów na mapie

3. Wykorzystane biblioteki zewnętrzne

- <SDL.h> - główna biblioteka odpowiedzialna za silnik gry. Jest odpowiedzalna za m. in.:
  - Tworzenie okna gry

- Zarządzanie renderowaniem obrazu
- Obsługa zdarzeń jak ruch myszki lub kliknięcie klawiszu
- Hitboxy
- Operacje na kolorach

- <SDL_tff.h> - jest to dodatek do głównej biblioteki SDL. Jest odpowiedzialna za:
  - Zamiana tekstu na teksture obrazu, co posłużyło do wyświetlenia menu, nazwy gracza oraz napisów końcowych

4. Wykorzystane biblioteki wewnętrzne

- **<iostream>** - obsługa urządzeń wejścia i wyjścia
- **<vector>** - modyfikacja listy obiektów;
- **<queue>** - obsługa stosu; kluczowa dla algorytmu BHS
- **<cmath>** - obsługa funkcji matematycznych takich jak abs() (wartość bezwzględna)
- **<random>** - służy do generowania losowych ilczb; obliczanie losowości spadania beczek
- **<string>** - obsługa ciągów znaków; tworzenie napisów końcowych
- **<sstream>, <iomanip>** - strumienie tekstowe oraz ich formatowanie; pozwala na łączenie tekstu z liczbami
- **<algorithm>** - zbiór gotowych funkcji; funkcja shuffle() miesza typy boosterów wprowadzając losowość
- **<map>** - słownik przechowujący pary klucz – wartość; w algorytmie BHS każdemu kafelkowi przypisywana jest wartość jego poprzednika

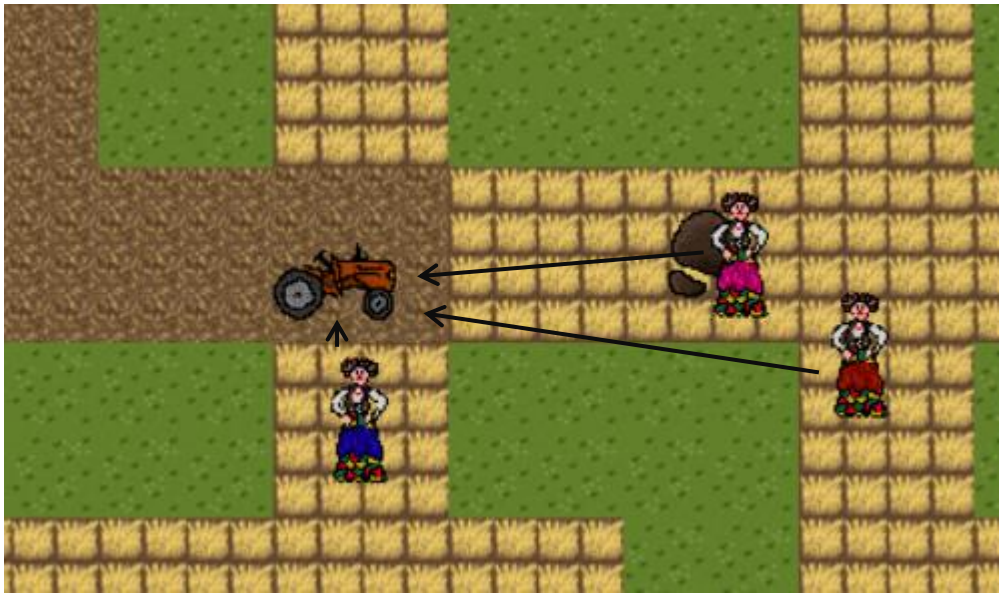
5. Wybrane funkcjonalności z kodem

1. Sztuczna Inteligencja Przeciwników (Algorytm BFS)

To zdecydowanie najbardziej zaawansowany element logiczny w kodzie. Zamiast poruszać się losowo, wrogowie używają algorytmu przeszukiwania wszerz, aby znaleźć najkrótszą ścieżkę do gracza w labiryncie.

```
void findPath(int startX, int startY, int targetX, int targetY) {
    path.clear();
    // Przeliczenie pikseli na współrzędne siatki (kolumny i wiersze)
    int sCol = startX / TILE_SIZE;
    int sRow = startY / TILE_SIZE;
    int tCol = targetX / TILE_SIZE;
    int tRow = targetY / TILE_SIZE;

    if (sCol == tCol && sRow == tRow) return; // Jeśli wróg jest na tym samym
polu co gracz, nie szukaj

    std::queue<SDL_Point> q;
    q.push({ sCol, sRow });

    std::map<int, SDL_Point> parentMap; // Mapa rodziców do odtworzenia ścieżki
    bool visited[MAP_ROWS][MAP_COLS] = { false };
    visited[sRow][sCol] = true;

    bool found = false;
    // Pętla algorytmu "zalewania" mapy
    while (!q.empty()) {
        SDL_Point curr = q.front(); q.pop();
```

```
        if (curr.x == tCol && curr.y == tRow) { found = true; break; }

        // Sprawdzanie sąsiadów (dół, góra, prawo, lewo)
        int dx[] = { 0, 0, 1, -1 };
        int dy[] = { 1, -1, 0, 0 };
        for (int i = 0; i < 4; i++) {
            int nx = curr.x + dx[i], ny = curr.y + dy[i];
            // Wróg może chodzić tylko po polach 0 (zboże) i 2 (ściernisko), nie
po 1 (ściana)
            if (nx >= 0 && nx < MAP_COLS && ny >= 0 && ny < MAP_ROWS &&
(maze[ny][nx] == 0 || maze[ny][nx] == 2) && !visited[ny][nx]) {
                visited[ny][nx] = true;
                parentMap[ny * MAP_COLS + nx] = curr;
                q.push({ nx, ny });
            }
        }
    }

    // Odtwarzanie ścieżki od końca (od gracza do wroga)
    if (found) {
        SDL_Point curr = { tCol, tRow };
        while (curr.x != sCol || curr.y != sRow) {
            // Dodajemy offset +10, żeby wróg celował w środek kafelka, a nie w
róg
            path.push_back({ curr.x * TILE_SIZE + 10, curr.y * TILE_SIZE + 10 });
            curr = parentMap[curr.y * MAP_COLS + curr.x];
        }
        std::reverse(path.begin(), path.end()); // Odwracamy ścieżkę, żeby była
od wroga do gracza
    }
}
```

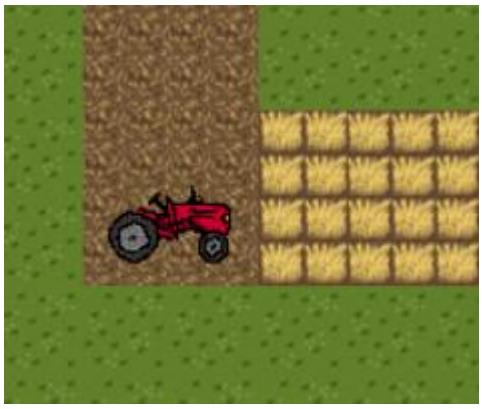


2. Fizyka Ruchu i Kolizje ze Ścianami

Podstawowa mechanika sterowania traktorem. Kluczowe jest tutaj to, że gra nie tylko przesuwa postać, ale sprawdza, czy nowy ruch nie wchodzi w ścianę.

```
void handleInput(SDL_Event& e) {
    if (e.type == SDL_KEYDOWN) {
        // Zapamiętujemy starą pozycję przed ruchem
        int oldX = rect.x;
        int oldY = rect.y;

        // Zmiana pozycji w zależności od wciśniętej strzałki
        switch (e.key.keysym.sym) {
```

```
        case SDLK_UP: rect.y -= currentSpeed; break;
        case SDLK_DOWN: rect.y += currentSpeed; break;
        case SDLK_LEFT: rect.x -= currentSpeed; break;
        case SDLK_RIGHT: rect.x += currentSpeed; break;
        }

        // Sprawdzanie kolizji ze ścianami mapy
        for (int r = 0; r < MAP_ROWS; r++) {
            for (int c = 0; c < MAP_COLS; c++) {
                if (maze[r][c] == 1) { // Jeśli pole jest ścianą (wartość 1)
                    // Tworzymy prostokąt ściany
                    SDL_Rect wall = { c * TILE_SIZE, r * TILE_SIZE, TILE_SIZE,
TILE_SIZE };

                    // Sprawdzamy czy prostokąt gracza nachodzi na prostokąt
ściany
                    if (SDL_HasIntersection(&rect, &wall)) {
                        // Jeśli tak, cofamy ruch (anulujemy zmianę pozycji)
                        rect.x = oldX;
                        rect.y = oldY;
                    }
                }
            }
        }
    }
}
```

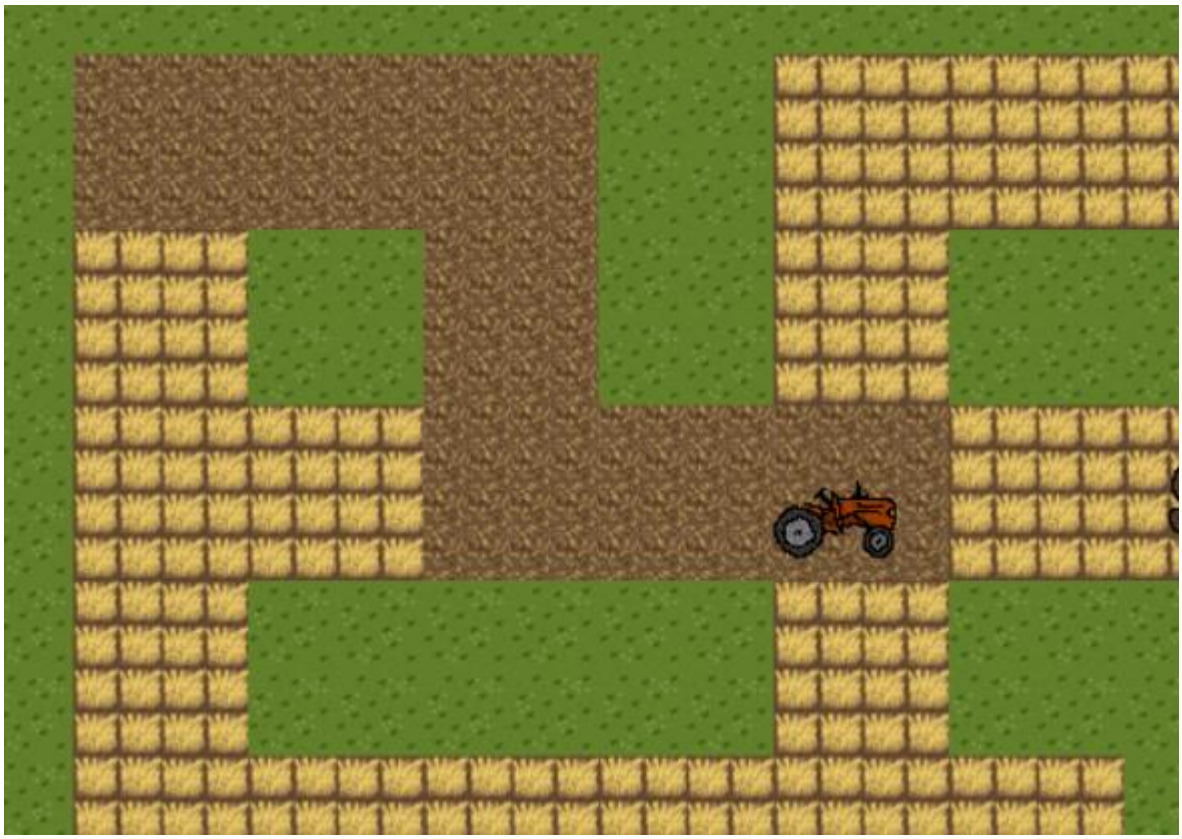


3. Logika "Koszenia" i Zwycięstwa

To mechanizm, który zmienia stan gry w czasie rzeczywistym. Kiedy gracz wchodzi na pole ze zbożem (0), zmienia się ono na ściernisko (2), a licznik punktów rośnie.

```
else {
    // Logika Zwykła: Koszenie zboża
    int pCol = (player.getX() + 25) / TILE_SIZE;
    int pRow = (player.getY() + 25) / TILE_SIZE;
    if (maze[pRow][pCol] == 0) {
        maze[pRow][pCol] = 2; // Zmiana na skoszone
        aktualnePunkty += 10;
        pozostalePunkty--;
    }
}


else if (maze[r][c] == 0) {
    if (cropTexture) SDL_RenderCopy(renderer, cropTexture, NULL, &tileRect);
    else { // Zapasowa kropka (zboże)
        SDL_SetRenderDrawColor(renderer, 255, 255, 255, 255);
        SDL_Rect dot = { c * TILE_SIZE + 32, r * TILE_SIZE + 32, 6, 6 };
        SDL_RenderFillRect(renderer, &dot);
    }
```

```
}
else if (maze[r][c] == 2) {
    if (stubbleTexture) SDL_RenderCopy(renderer, stubbleTexture, NULL,
&tileRect);
    else { // Zapasowy kolor ziemi
        SDL_SetRenderDrawColor(renderer, 139, 69, 19, 255);
        SDL_RenderFillRect(renderer, &tileRect);
    }
}
```



4. System Zapisu i Odczytu Postępu (Persistence)

Funkcjonalność pozwalająca na zachowanie stanu odblokowanych poziomów po wyłączeniu gry. Używa standardowej biblioteki fstream.

```
// Funkcja zapisu postępu do pliku tekstowego (0 = zablokowany, 1 = odblokowany)
void zapiszPostep(bool odblokowane[], int rozmiar) {
    std::ofstream plik("postep.txt");
    if (plik.is_open()) {
        for (int i = 0; i < rozmiar; i++) {
            plik << odblokowane[i] << " ";
        }
        plik.close();
    }
}

// Funkcja odczytu postępu. Jeśli plik nie istnieje, tworzy domyślny stan.
void wczytajPostep(bool odblokowane[], int rozmiar) {
    std::ifstream plik("postep.txt");
    if (plik.is_open()) {
        for (int i = 0; i < rozmiar; i++) {
            plik >> odblokowane[i];
        }
        plik.close();
    }
    else {
```

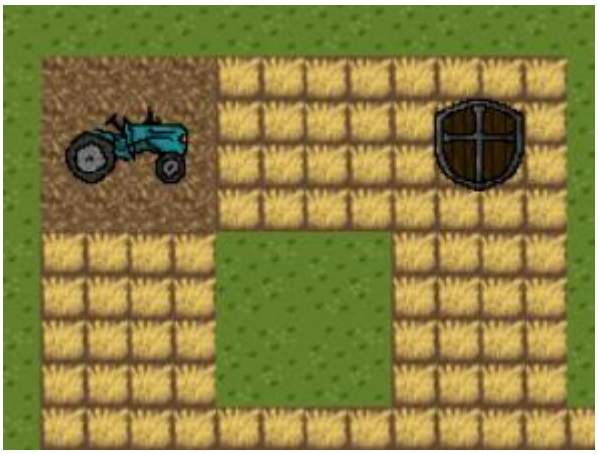
```
        // Domyślnie odblokowany tylko 1 poziom, reszta zablokowana
        odblokowane[0] = true;
        for (int i = 1; i < rozmiar; i++) odblokowane[i] = false;
    }
}
```

5. Wybór Koloru Gracza (Personalizacja UI)

System interfejsu w menu, umożliwiający personalizację postaci. Gra przechowuje listę dostępnych wariantów (tekstura + kolor RGB) w wektorze struktur. Obsługa myszy wykrywa kliknięcie w kolorowy przycisk, zapamiętuje wybór użytkownika, a przy starcie poziomu dynamicznie podmienia teksturę obiektu gracza na wybraną.

```
// Sprawdzenie czy kliknięto w wybór koloru
for (int i = 0; i < 8; i++) {
    if (SDL_PointInRect(&mousePos, &colorButtons[i])) {
        selectedColorIndex = i;
    }
}
```

```
player.setTexture(tractorOptions[selectedColorIndex].texture);
```



6. README

---README---
Aby uruchomić grę należy mieć na urządzeniu zainstalowane środowisko programistyczne Visual Studio.
Należy pobrać plik zip repozytorium i odpakować w dogodnym miejscu. Następnie otworzyć plik TractorMazeRunner/TractorMazeRunner.sln w Visual Studio.
Pliki bibliotek i assetow znajdują się już w repozytorium, gra nie wymaga instalacji dodatkowych plików.

---OPIS GRY---
Fabuła:
Po zeszłorocznych dożynkach na dotychczas spokojnych wsiach wybucha konflikt. Sołtys Suchej Psiny postanawia sabotować udział wsi Zimna Wódka w konkursie na najlepszą koronę dożynkową. W tym celu zwołuje Koło Gospodyń Wiejskich (KGW) i razem wyruszają na pola w Zimnej Wódce, aby przeszkodzić w zbiorach i przygotowywaniu korony.

Cel gry:
Gracz wciela się w role rolnika z Zimnej Wódki, przechodzi poziomy zbierając plony i korony, jednocześnie uciekając przed zawistnymi babami i sołtysem Suchej Psiny.
W menu może wybrać własne imię, kolor traktora. W grze mamy dwa levele - 1 i 2 plus boss fight(3). Gracz może uruchomić kolejny dopiero po przejściu poprzedniego. W pierwszych dwóch należy zebrać całe zboże, uciekając przed kobietami. W boss fighcie trzeba zebrać wszystkie korony dożynkowe nie dając się zlapac sołtysowi i zgnieść balami slomy.

7. Link do Repozytorium

https://github.com/kard89/TractorMazeRunner