# GENERATOR POZIOMÓW DO GRY TYPU TOWER DEFENSE Z WYKORZYSTANIEM ALGORYTMU GENETYCZNEGO

european student magazine
Numer 1 / 2025

Maciej **Kwieciński**[1], Aleksandra **Sikora**[2*]

[1]Politechnika Świętokrzyska, Wydział Elektrotechniki, Automatyki i Informatyki, al. Tysiąclecia Państwa Polskiego 7, 25-314 Kielce, Polska, s092751@student.tu.kielce.pl
[2]Politechnika Świętokrzyska, Wydział Elektrotechniki, Automatyki i Informatyki, al. Tysiąclecia Państwa Polskiego 7, 25-314 Kielce, Polska, asikora@tu.kielce.pl
[*]Corresponding author

**Streszczenie** - Przedstawiony artykuł koncentruje się na implementacji aplikacji stanowiącej grę typu Tower Defense (TD). Aplikacja, poza standardowymi mechanikami gatunku, implementuje innowacyjne podejście do generowania poziomów poprzez zastosowanie algorytmu genetycznego. Przeprowadzone badania wykazały, iż wykorzystanie algorytmu genetycznego pozwala na znaczące rozszerzenie puli dostępnych map, zwiększając tym samym różnorodność rozgrywki. Implementacja aplikacji została zrealizowana w oparciu o silnik Unity, z wykorzystaniem języka C#.

**Słowa kluczowe -** gry komputerowe, unity, algorytmy genetyczne, generowanie poziomów

## Wstęp

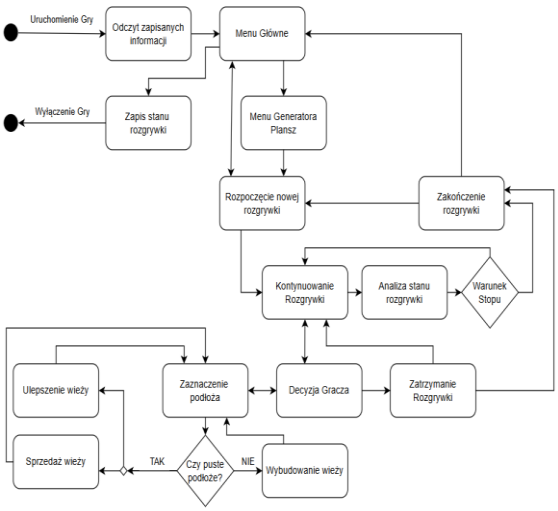
Typowa mechanika gry Tower Defense (TD) opiera się na poruszaniu się wrogów po z góry ustalonej trasie, a zadaniem gracza jest ich eliminacja za pomocą wież. Za zniszczenie każdej jednostki gracz otrzymuje zasoby, które może przeznaczyć na budowę nowych, potężniejszych struktur obronnych lub ulepszanie już istniejących. Zwycięstwo w grze osiągane jest poprzez skuteczne odparcie wszystkich fal wrogów, natomiast porażka następuje w momencie, gdy określona liczba przeciwników dotrze do bronionego punktu.

Kluczowym elementem wpływającym na strategię rozgrywki w tych grach jest różnorodność map. Układ ścieżek, rozmieszczenie strategicznych punktów oraz ukształtowanie terenu mają fundamentalne znaczenie przy doborze odpowiednich wież i ich optymalnym rozmieszczeniu. Zróżnicowanie map pod względem długości i krętości tras wpływa na preferowane strategie – mapy z długimi, krętymi ścieżkami sprzyjają wykorzystaniu wież o dużym zasięgu, podczas gdy mapy z krótkimi, prostymi odcinkami wymagają skupienia na sile ognia. Jednakże, powszechnym problemem w tradycyjnych grach TD jest ograniczona liczba dostępnych map. Ten brak różnorodności prowadzi do szybkiego znużenia graczy, którzy po krótkim czasie zapamiętują dostępne układy plansz, co znacząco obniża regrywalność i skraca żywotność tytułu. Brak dynamicznej generacji map stanowi zatem istotne ograniczenie dla rozwoju gatunku.

## I. PROJEKT I ARCHITEKTURA GRY

Niniejsza gra reprezentuje gatunek Tower Defense, wzbogacony o mechanizm losowej generacji map. Zadaniem gracza jest strategiczne rozmieszczanie i rozbudowa wież obronnych wzdłuż wyznaczonej trasy, którą przemieszczają się fale przeciwników. Głównym celem rozgrywki jest ochrona bazy przed zniszczeniem poprzez eliminację nacierających jednostek wroga.

Diagram stanów rozgrywki (Rys. 1.) przedstawia w sposób graficzny przepływ sterowania i logiki w grze, ilustrując możliwe stany, w jakich może znajdować się gra. Diagram przedstawia również przejścia między kolejnymi stanami w odpowiedzi na akcje gracza lub zdarzenia w grze.




**Rys. 1. Diagram stanów rozgrywki**

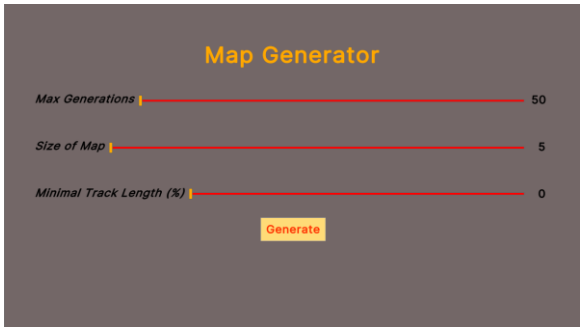
## II. GRAFICZNY INTERFEJS UŻYTKOWNIKA

Interfejs graficzny użytkownika (GUI) w grze został stworzony w Unity[7] z użyciem UI Buildera i UXML, co umożliwiło stylowanie elementów interfejsu na wzór CSS. Dynamiczne zarządzanie widocznością komponentów zapewnia graczowi dostęp tylko do istotnych w danym momencie ekranów i informacji.

Menu główne (Rys. 2) zawiera cztery centralnie umieszczone przyciski funkcyjne: „PLAY” (inicjuje nową rozgrywkę), „Generate Custom Map” (dostęp do opcji generowania map), „Exit” (zamyka aplikację).

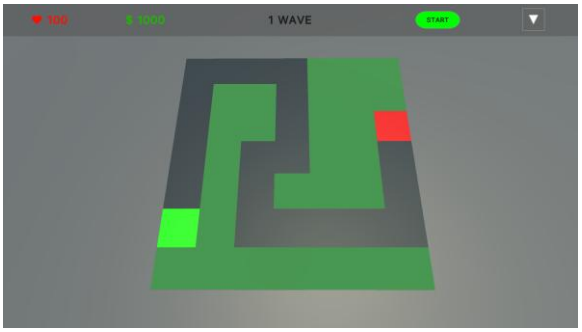

**Rys. 2. Zrzut ekranu menu głównego**

Panel generatora map (Rys. 3) służy do konfigurowania parametrów generowanych plansz i prezentuje trzy suwaki: „Max Generations” (kontroluje maksymalną liczbę iteracji algorytmu), „Size of Map” i „Minimal Track Length (%)” (wpływają na wartość poszukiwanej optymalnej długości trasy). Opcja „Generate” inicjuje proces generowania.



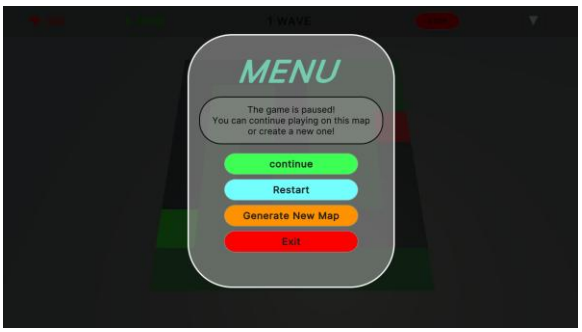

**Rys. 3. Zrzut ekranu menu generatora map**

Interfejs rozgrywki (Rys. 4.) prezentuje aktualny stan gry. Centralną część ekranu zajmuje plansza, ukazana w rzucie z góry, przedstawiająca ścieżkę, po której poruszają się przeciwnicy. Na planszy wyróżnione są dwa kluczowe punkty: startowy (zielony) i docelowy (czerwony). W górnej części ekranu znajdują się wskaźniki: życia gracza, waluty oraz fali. W prawym górnym rogu znajduje się przycisk „START”, inicjujący kolejną falę. Strzałka skierowana w dół uruchamia menu pauzy.




**Rys. 4. Zrzut ekranu interfejsu podczas rozgrywki**

Menu pauzy/podsumowania rozgrywki (Rys. 5) pojawia się po wstrzymaniu rozgrywki lub po zakończeniu poziomu. Jego zawartość dynamicznie dostosowuje się do aktualnego stanu gry, udostępniając takie opcje, jak: kontynuowanie i restartowanie rozgrywki, przejście do generatora map lub wyjście do menu głównego.

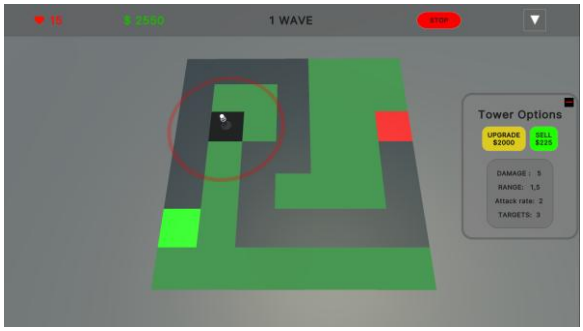



**Rys. 5. Zrzut ekranu menu pauzy**

Panel wyboru wieży (Rys. 6) umożliwia graczowi wybór i budowę wież obronnych podczas rozgrywki. Panel ten pojawia się po naciśnięciu myszką na dostępne podłoże na planszy i zawiera listę dostępnych typów wież wraz z ich cenami.


**Rys. 6. Zrzut ekranu menu wyboru wieży**

Panel zarządzania wybudowaną wieżą (Rys. 7) pojawia się po kliknięciu myszką na istniejącą wieżę na planszy. Umożliwia on graczowi zarządzanie wybraną wieżą, oferując opcje ulepszenia i sprzedaży. Panel ten wyświetla również statystyki wybranej wieży, takie jak zadawane obrażenia, zasięg, szybkostrzelność i ilość celów, które może jednocześnie atakować.

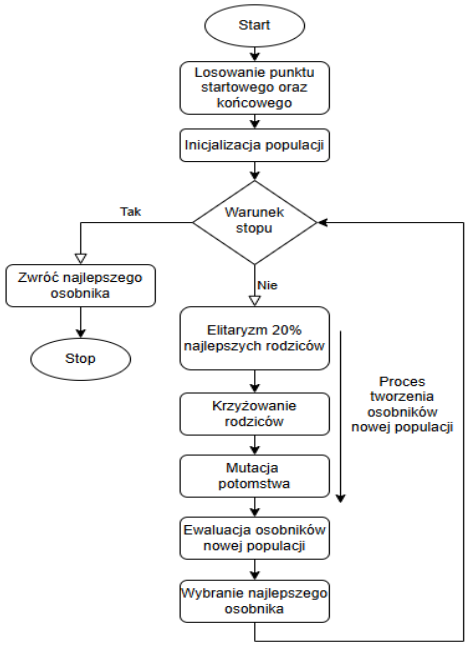



**Rys. 7. Zrzut ekranu menu zarządzania wieżą**

## III. ALGORYTM GENETYCZNY

Algorytmy genetyczne[1], jako najliczniejsza grupa algorytmów ewolucyjnych, znajdują szerokie zastosowanie w badaniach nad grami, w tym w problemach takich jak generowanie i balansowanie fal przeciwników [3], optymalne rozmieszczenie jednostek [6] czy proceduralne generowanie poziomów [5]. W prezentowanym rozwiązaniu algorytm genetyczny został wykorzystany do automatycznego generowania nowych poziomów, co ma na celu wydłużenie i urozmaicenie rozgrywki.

Schemat blokowy ogólnego przebiegu działania wykorzystanego algorytmu genetycznego przedstawiono na Rysunku 8.



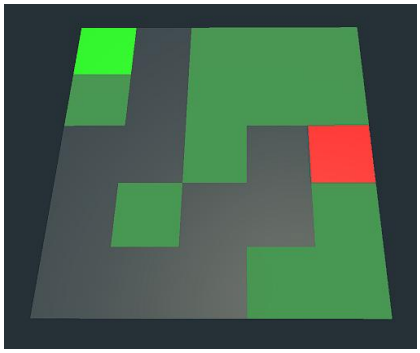

**Rys. 8 Schemat blokowy algorytmu genetycznego**

### Implementacja algorytmu

Proces działania algorytmu genetycznego rozpoczyna się od wyboru punktu startowego oraz końcowego, a następnie pseudolosowej inicjalizacji populacji początkowej, w której każdy osobnik jest reprezentowany jako macierz dwuwymiarowa, o z góry zdefiniowanej wielkości. Macierz ta przedstawia abstrakcyjny model mapy gry, gdzie każda kolejna komórka określa jej przynależność do pustego obszaru oznaczonego stanem “0” który może zostać wybrany jako część ścieżki lub obszaru zablokowanego oznaczonego stanem “1”. Miejsce startu oraz końca oznaczone jest stanem o numerze “3”. Przedstawiony został przykład wygenerowanej planszy w postaci macierzy (Rys. 9) oraz jej wygenerowany model w grze (Rys. 10).

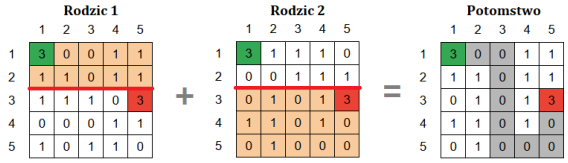
| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 3 | 0 | 1 | 1 | 1 |
| 2 | 1 | 0 | 1 | 1 | 1 |
| 3 | 0 | 0 | 1 | 0 | 3 |
| 4 | 0 | 1 | 0 | 0 | 1 |
| 5 | 0 | 0 | 0 | 1 | 1 |

**Rys. 9. Abstrakcyjny przykład macierzy wygenerowanej mapy 5x5**



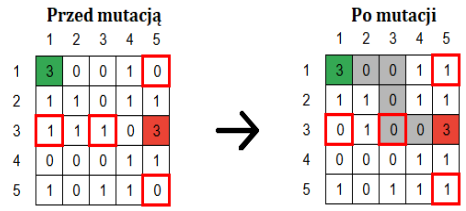
**Rys. 10. Przykład wygenerowanego modelu mapy 5x5**

Na początku każdej generacji algorytm przeprowadza wstępną selekcję zgodną z zasadą elitaryzmu[2], gdzie ustalony % najlepiej przystosowanych osobników z poprzedniej generacji przechodzi do puli osobników nowej populacji, a następnie za pomocą generatora liczb pseudolosowych, losuje kolejne pary osobników z populacji. W procesie krzyżowania dwa wybrane osobniki tworzą nowego potomka, który łączy cechy rodziców korzystając z tak zwanego punktu krzyżowania (Rys. 11).




**Rys. 11. Abstrakcyjny przykład krzyżowania jednopunktowego**

Z niskim prawdopodobieństwem każdy potomek może ulec mutacji, która wprowadza niewielkie, losowe zmiany w jego macierzy (Rys. 12). Mutacje te pozwalają na eksplorację nowych obszarów poszukiwań.




**Rys. 12. Abstrakcyjny przykład działania mutacji**

Po każdej generacji każdy osobnik nowej populacji jest oceniany za pomocą funkcji przystosowania. W prezentowanym rozwiązaniu funkcja ta wykorzystuje algorytm A* [4], który przeszukuje każdego osobnika w poszukiwaniu możliwej trasy od punktu startowego do końcowego. Długość znalezionej trasy staje się miarą przystosowania osobnika. W przypadku braku możliwej trasy, osobnik otrzymuje wartość 0. Algorytm zatrzymuje się, gdy zostanie osiągnięta maksymalna liczba generacji lub gdy zostanie wyłoniony osobnik o długości trasy nie mniejszej niż wyznaczona optymalna długość trasy.

Wielkość A przedstawia funkcję wyznaczającą optymalną długość trasy :

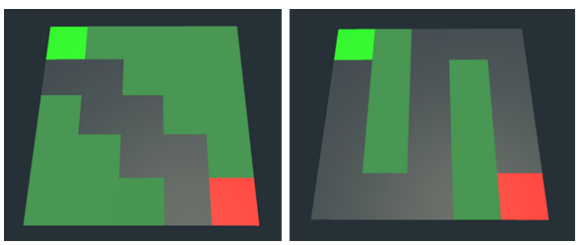
$$A(x,y) = (2x-2)(1+\frac{y}{100}(\lfloor\frac{x}{4}\rfloor-1)) \qquad (1)$$

gdzie:

x - parametr oznaczający rozmiar generowanej mapy

y - parametr oznaczający stosunek wypełnienia mapy trasą

Przykłady wygenerowanych map, oparte na obliczeniach ze wzoru, prezentuje Rysunek 13. Ilustruje on mapy o najmniejszej i największej możliwej długości trasy, dla rozmiaru mapy równej 5 oraz współczynnika jej wypełnienia równym 0 oraz 100.



**Rys. 13. Przykład wygenerowanej mapy o najmniejszej i największej możliwej długości trasy, dla planszy o wielkości 5x5**

## IV. Wpływ doboru parametrów algorytmu genetycznego

Działanie algorytmu genetycznego, opisanego w poprzednich podrozdziałach, jest silnie uzależnione od doboru odpowiednich parametrów. Parametry te determinują zarówno czas potrzebny do znalezienia satysfakcjonującego rozwiązania, jak i jakość generowanych map, mierzoną długością wyznaczonej trasy. W niniejszym podrozdziale przeanalizowano wpływ następujących parametrów na proces generacji map: rozmiar populacji, współczynnik krzyżowania, mutacji oraz elitaryzmu.

### Parametry testów

W celu zbadania wpływu poszczególnych parametrów zastosowano iteracyjne podejście optymalizacyjne. Proces ten polegał na testowaniu każdego parametru oddzielnie, przy zachowaniu stałych wartości pozostałych parametrów. Po znalezieniu optymalnej wartości danego parametru, była ona wykorzystywana jako wartość bazowa w kolejnych testach, optymalizujących następny parametr. Wartości początkowe parametrów, użyte w pierwszym etapie testów (optymalizacja rozmiaru populacji), przedstawiono w Tabeli 1, natomiast Tabela 2 przedstawia stałe wartości.

**Tabela 1 Wartości początkowe i testowe parametrów**

| Parametr | Wartość bazowa | Testowane wartości |
|---|---|---|
| Rozmiar populacji | 300 | 100, 200, 300, 500 |
| Współczynnik krzyżowania | 0.7 | 0.6, 0.7, 0.8, 0.9 |
| Współczynnik mutacji | 0.01 | 0.01, 0.02, 0.03, 0.05, 0.07, 0.1 |
| Elitaryzm | 20% | 10%, 20%, 30%, 40%, 50% |

**Tabela 2 Stałe wartości wykorzystane podczas testów**

| Parametr | Wartość |
|---|---|
| Liczba testów | 100 |
| Maksymalna liczba generacji | 300 |
| Rozmiar mapy | 7x7 |
| Stosunek wypełnienia mapy trasą | 60 |
| Optymalna długość trasy | 19.2 |

W celu zbadania wpływu poszczególnych parametrów zastosowano iteracyjne podejście optymalizacyjne. Proces ten polegał na testowaniu każdego parametru oddzielnie, przy zachowaniu stałych wartości pozostałych parametrów. Po znalezieniu optymalnej wartości danego parametru, była ona wykorzystywana jako wartość bazowa w kolejnych testach, optymalizujących następny parametr.

### Wpływ rozmiaru populacji

Niniejszy podrozdział analizuje wpływ wielkości populacji na efektywność algorytmu genetycznego w kontekście generowania map. Przeprowadzono serię eksperymentów dla populacji o liczebnościach: 100, 200, 300 i 500 osobników. Pozostałe parametry algorytmu genetycznego zostały ustalone zgodnie z wartościami przedstawionymi w Tabeli 2.

Wyniki testów przedstawiono w Tabeli 3, uwzględniając metryki takie jak średni czas wykonania algorytmu, średnia wartość funkcji przystosowania, liczba unikalnych i kompletnych ścieżek wygenerowanych w populacji oraz liczba iteracji potrzebnych do znalezienia rozwiązania spełniającego kryterium optymalnej długości trasy.

**Tabela 3 Wyniki testów dla różnych rozmiarów populacji**

| Rozmiar populacji | Średni czas wykonania | Średnia wartość przystosowania | Średnia liczba iteracji |
|---|---|---|---|
| 100 | 303 | 100 | 35 |
| 200 | 312 | 86 | 20 |
| 300 | 369 | 82 | 17 |
| 500 | 548 | 77 | 15 |

Dane wskazują, że czas wykonania (znalezienia ścieżki) znacząco rośnie wraz ze wzrostem liczby osobników. Dla populacji 100 osobników uzyskano najkrótszy czas (303 ms), podczas gdy dla populacji liczącej 500 osobników czas ten zwiększył się do 548 ms. Analiza danych wykazała również, że wraz ze wzrostem liczebności populacji jakość generowanych rozwiązań maleje. Najniższą średnią wartość przystosowania (77) odnotowano dla populacji liczącej 500 osobników, co sugeruje, że mniejsze populacje generują bardziej optymalne rozwiązania. Populacja o wielkości 200 osobników zapewnia optymalny balans między czasem obliczeń a jakością uzyskanych wyników.

### Wpływ współczynnika krzyżowania

W tej sekcji omówiono wpływ różnych współczynników krzyżowania (0.6, 0.7, 0.8 i 0.9) na działanie algorytmu genetycznego. Wielkość populacji ustalono na 200 osobników, a pozostałe parametry przyjęto zgodnie z Tabelą 3. Wyniki testów zestawiono w Tabeli 4.

**Tabela 4 Wyniki testów dla różnych współczynników krzyżowania**

| Współczynnik krzyżowania | Średni czas wykonania | Średnia wartość przystosowania | Średnia liczba iteracji |
|---|---|---|---|
| 0.6 | 333 | 103 | 36 |
| 0.7 | 301 | 100 | 36 |
| 0.8 | 289 | 96 | 34 |
| 0.9 | 282 | 95 | 31 |

Analiza czasowa wykonania algorytmu w zależności od współczynnika krzyżowania wykazała, że czas ten maleje wraz z jego wzrostem. Przy współczynniku równym 0.6 czas wykonania wyniósł 333 ms, natomiast przy 0.9 skrócił się do 282 ms. Badania wykazały również, że wraz ze wzrostem współczynnika jakość wyników nieznacznie spada. Jednak zmiany te są relatywnie niewielkie w porównaniu z korzyściami czasowymi. W przeprowadzonych testach współczynnik 0.9 okazał się korzystnym kompromisem, ponieważ zmiany w jakości rozwiązań są relatywnie niewielkie w porównaniu ze zmianami w czasie wykonania.

### Wpływ współczynnika mutacji

W tym podrozdziale przeanalizowany został wpływ współczynnika mutacji na działanie algorytmu. Eksperymenty przeprowadzono dla wartości: 0.01, 0.02, 0.03, 0.05, 0.07 i 0.1. Pozostałe parametry algorytmu przyjęto zgodnie z wcześniejszymi ustaleniami: populacja 200 osobników, współczynnik krzyżowania 0.9. Wyniki testów przedstawiono w Tabeli 5.

**Tabela 5 Wyniki testów dla różnych współczynników mutacji**

| Współczynnik mutacji | Średni czas wykonania | Średnia wartość przystosowania | Średnia liczba iteracji |
|---|---|---|---|
| 0.01 | 279 | 95 | 33 |
| 0.02 | 241 | 81 | 31 |
| 0.03 | 208 | 70 | 29 |
| 0.05 | 215 | 59 | 31 |
| 0.07 | 241 | 55 | 39 |
| 0.1 | 314 | 50 | 49 |

Analiza wpływu współczynnika mutacji na średni czas wykonania algorytmu wykazała, że czas ten początkowo maleje wraz ze wzrostem wartości współczynnika mutacji, osiągając minimum dla wartości 0.03, po czym zaczyna rosnąć. Badanie zależności średniej wartości funkcji przystosowania ukazują, że zwiększanie wartości tego parametru prowadzi do stopniowego pogorszenia jakości rozwiązań. Wyniki wskazują na fakt, że zależność ta nie jest monotoniczna i istnieje optymalny zakres tej wartości. W przypadku omawianych testów, współczynnik mutacji w okolicach wartości 0.03 oferuje korzystny kompromis między czasem wykonania a jakością uzyskiwanych rozwiązań..

### Wpływ współczynnika elitaryzmu

Niniejszy podrozdział analizuje wpływ współczynnika elitaryzmu na algorytm genetyczny. Testy przeprowadzono dla wartości: 10%, 20%, 30%, 40% i 50%. Przyjęto populację liczącą 200 osobników, współczynnik krzyżowania 0.9 oraz współczynnik mutacji 0.03. Wyniki przeprowadzonych testów

przedstawia tabela 6.

**Tabela 6. Wyniki testów dla różnych współczynników elitaryzmu**

| Współczynnik elitaryzmu | Średni czas wykonania | Średnia wartość przystosowania | Średnia liczba iteracji |
|---|---|---|---|
| 10 % | 199 | 47 | 33 |
| 20 % | 203 | 70 | 28 |
| 30 % | 250 | 78 | 28 |
| 40 % | 270 | 105 | 33 |
| 50 % | 326 | 118 | 38 |

Analiza wpływu współczynnika elitaryzmu na czas wykonania algorytmu wykazała, że czas ten wzrasta wraz z wartością współczynnika. Najkrótszy czas zaobserwowano dla elitaryzmu równego 10%, a najdłuższy dla 50%. Natomiast, w przeciwieństwie do czasu wykonania, jakość rozwiązań poprawia się wraz ze wzrostem współczynnika elitaryzmu. Zwiększając więc ten współczynnik, algorytm potrzebuje więcej czasu na obliczenia, ale w zamian generuje lepsze wyniki. W przeprowadzonych testach, współczynnik elitaryzmu w przedziale 30-40% wydaje się oferować optymalny kompromis między czasem wykonania a jakością rozwiązań.

## V. TESTY WYDAJNOŚCIOWE

Testy zostały przeprowadzone na platformie testowej o następującej specyfikacji:

**Procesor (CPU):** Intel Core i5-4460 (4 rdzenie, 3.2 GHz)
**Pamięć RAM:** 16 GB DDR3 1600MHz
**Karta Graficzna (GPU):** AMD Radeon R9 380x
**System Operacyjny:** Windows 10 64-bit

### Testy wydajnościowe mechaniki gry

W ramach przeprowadzonych badań, mających na celu ocenę charakterystyk wydajnościowych implementowanej mechaniki rozgrywki, zidentyfikowano obszar potencjalnej optymalizacji w obrębie mechanizmów odpowiedzialnych za działanie wież obronnych oraz jednostek przeciwników. Wstępna analiza profilowania wykazała, iż znaczący wpływ na obciążenie procesora miało dynamiczne generowanie instancji obiektów (pocisków) przy każdym cyklu ataku wieży, połączone z instancjonowaniem efektów wizualnych reprezentujących obrażenia zadawane jednostkom przeciwnika. W celu minimalizacji kosztów związanych z dynamiczną alokacją i dealokacją pamięci, zaimplementowano wzorzec projektowy „Object Pooling". Zastosowana metoda polega na jednorazowej inicjalizacji puli obiektów, a następnie ich cyklicznej aktywacji i dezaktywacji, co pozwala na ponowne wykorzystanie istniejących instancji, znacząco redukując narzut związany z zarządzaniem pamięcią.

Szczególną uwagę poświęcono analizie wydajności wież typu Tesla, charakteryzujących się zdolnością do jednoczesnego atakowania do trzech celów (lub sześciu po ulepszeniu), a także generowaniem złożonych efektów wizualnych. Cechy te czynią je najbardziej zasobożernymi elementami spośród dostępnych struktur obronnych. W celu przeprowadzenia testów obciążeniowych, zaaranżowano scenariusz z udziałem 50 wież typu Tesla oraz 300 jednostek przeciwników. W warunkach maksymalnego obciążenia, tj. jednoczesnego ataku wszystkich wież, zaobserwowano spadek wartości wskaźnika liczby klatek na sekundę (FPS). Niemniej jednak, wartość ta utrzymywała się powyżej progu 60 FPS, co uznaje się za wartość zapewniającą płynną rozgrywkę. Szczegółowe pomiary wykazały, że czas realizacji wszystkich ataków i związanych z nimi efektów wizualnych wyniósł średnio 11,5 ms. Całkowity czas generowania pojedynczej klatki w warunkach maksymalnego obciążenia wyniósł 14,5 ms, co odpowiada średniej wartości FPS równej liczbie 69. Uzyskane wyniki wskazują, że implementacja wzorca „Object Pooling" pozwoliła na osiągnięcie satysfakcjonującej wydajności, umożliwiając płynną rozgrywkę na platformie testowej oraz na platformach o wyższej specyfikacji, nawet w warunkach znacznego obciążenia systemu. W standardowych warunkach rozgrywki, tj. przy mniejszym obciążeniu systemu, zmierzony czas generowania pojedynczej klatki mieścił się w przedziale od 3 do 5 ms, co przekłada się na uzyskanie od 200 do 300 klatek na sekundę.

### Testy wydajnościowe generatora map

W celu oceny wydajności algorytmu generowania map, przeprowadzono serię testów, których wyniki przedstawiono w Tabeli 8, natomiast parametry wykorzystane podczas testów znajdują się w Tabeli 7. Testy te miały na celu zbadanie zależności pomiędzy parametrami wejściowymi generatora a charakterystykami generowanych map, takimi jak średni czas znajdowania trasy, średnia długość znalezionej trasy oraz długość trasy optymalnej. (Długość optymalnej trasy obliczana jest zgodnie ze wzorem (1), gdzie X oznacza rozmiar mapy, a Y – współczynnik jej wypełnienia trasą)

**Tabela 7 Lista parametrów wykorzystanych podczas testów wydajnościowych**

| Parametr | Wartość |
|---|---|
| Maksymalna liczba generacji algorytmu genetycznego | 300 |
| Liczba przeprowadzonych testów | 10 |

**Tabela 8 Analiza średniego czasu wygenerowania tras w odniesieniu do wymagań użytkownika**

| X \ Y | 0 | 20 | 40 | 60 | 80 | 100 |
|---|---|---|---|---|---|---|
| 5 | 281 ms | 279 ms | 342 ms | 335 ms | 929 ms | 1140 ms |
| 6 | 407 ms | 432 ms | 434 ms | 416 ms | 645 ms | 580 ms |
| 7 | 623 ms | 577 ms | 510 ms | 517 ms | 712 ms | 739 ms |
| 8 | 617 ms | 642 ms | 631 ms | 674 ms | 638 ms | 859 ms |
| 9 | 812 ms | 841 ms | 887 ms | 1664 ms | 4111 ms | 4372 ms |
| 10 | 952 ms | 958 ms | 992 ms | 2154 ms | 4554 ms | 5779 ms |
| 11 | 1217 ms | 1272 ms | 1235 ms | 2469 ms | 5625 ms | 6571 ms |
| 12 | 1384 ms | 1387 ms | 1638 ms | 3616 ms | 6973 ms | 8435 ms |
| 13 | 1596 ms | 1669 ms | 3563 ms | 9200 ms | 9836 ms | 10034 ms |
| 14 | 1735 ms | 1778 ms | 4684 ms | 11131 ms | 11928 ms | 11994 ms |
| 15 | 1794 ms | 1754 ms | 5332 ms | 11815 ms | 12518 ms | 13258 ms |

Każdy test był średnią arytmetyczną z 10 niezależnych prób, gdzie każda próba obejmowała czyszczenie poprzedniej mapy, generowanie nowej i tworzenie jej wpełni funkcjonalnego modelu graficznego.

Analiza danych z Tabeli 8 pokazuje istotne zależności pomiędzy rozmiarem generowanej mapy (X), stopniem jej wypełnienia trasą (Y) a efektywnością procesu generowania. Obserwuje się wyraźny wzrost czasu generowania map wraz ze zwiększaniem wartości X, co jest zgodne z oczekiwaniami, biorąc pod uwagę złożoność obliczeniową algorytmu w kontekście rosnącej przestrzeni poszukiwań. Wpływ parametru Y na czas generowania nie jest liniowy. Wyraźnie zaznacza się tendencja wzrostowa czasu generowania wraz ze zwiększaniem wartości Y, ze szczególnym uwzględnieniem map o rozległych wymiarach.

Szczególnie istotnym aspektem, który należy uwzględnić w interpretacji wyników, jest fakt, że dla wysokich wartości Y, a w szczególności dla Y=100, liczba możliwych tras spełniających kryterium wypełnienia mapy ulega znacznemu ograniczeniu. W skrajnym przypadku, dla Y=100, istnieje zazwyczaj tylko jedno unikalne rozwiązanie. To istotnie zawęża przestrzeń poszukiwań, ale jednocześnie drastycznie zwiększa trudność znalezienia tego konkretnego rozwiązania, co bezpośrednio przekłada się na wydłużenie czasu generowania map. Zatem obserwowane wydłużenie czasu generowania dla wysokich wartości Y niekoniecznie musi wynikać z samej złożoności algorytmu, ale również z ograniczonej liczby akceptowalnych rozwiązań.

## VI. Wnioski

W niniejszej pracy podjęto problematykę dynamicznej generacji map w grach z gatunku Tower Defense, co stanowi istotne rozszerzenie możliwości rozgrywki poprzez zapewnienie niemal nieograniczonej liczby unikalnych konfiguracji plansz. Opracowano prototypową aplikację, wykorzystując środowisko Unity oraz język C#, które okazały się adekwatne do realizacji zdefiniowanych celów, umożliwiając efektywną implementację i walidację proponowanego rozwiązania.

Przedstawiona aplikacja implementuje standardowe mechanizmy rozgrywki Tower Defense, polegające na obronie strategicznej trasy przed nadciągającymi falami przeciwników poprzez rozmieszczanie, modyfikację (w tym sprzedaż i ulepszanie) struktur obronnych (wież). Wdrożono również mechanizm progresywnego wzrostu trudności przeciwników wraz z postępem rozgrywki (kolejne fale). Kluczowym elementem pracy jest implementacja algorytmu genetycznego, który został adaptowany i parametryzowany w celu optymalizacji procesu generowania map pod kątem szybkości i efektywności, z uwzględnieniem preferencji użytkownika.

Przeprowadzone testy wydajnościowe gry wykazały, że zastosowanie wzorca projektowego „Object Pooling" znacząco zwiększyło ilość generowanych klatek. Natomiast testy związane z generatorem plansz ujawniły znaczące zależności czasu ich generowania wraz ze wzrostem jej rozmiaru, co jest zgodne z oczekiwaniami. Wpływ parametru Y (stopień wypełnienia planszy) był bardziej złożony, z tendencją do wzrostu czasu generowania przy wyższych wartościach Y, zwłaszcza dla większych map. Szczególną uwagę zwrócono na fakt, że dla Y=100 liczba możliwych tras ulega znacznemu ograniczeniu, co istotnie wpływa na czas poszukiwania rozwiązania. W konsekwencji, wydłużenie czasu generowania dla wysokich wartości Y wynika nie tylko ze złożoności algorytmu, ale również z ograniczonej liczby akceptowalnych rozwiązań.

## DESIGN OF A LEVEL GENERATOR FOR A TOWER DEFENSE GAME WITH THE USE OF GENETIC ALGORITHM

The presented article focuses on the implementation of an application that is a Tower Defense game. The application, in addition to the standard mechanics of the species, implements an innovative approach to level generation through the use of a genetic algorithm. The research has shown that the use of a genetic algorithm allows for a significant expansion of the pool of available maps, thus increasing the variety of gameplay. The implementation of the application was based on the Unity engine, using the C# language.

**Key words**: pc games, unity, genetic algorithms, level generation

## BIBLIOGRAFIA

[1] Agoston E. Eiben and James E. Smith. 2015. Introduction to Evolutionary Computing (2nd. ed.). Springer Publishing Company. s. 25-100

[2] Du, Haiming & Wang, Zaichao & Zhan, Wei & Guo, Jinyi. (2018). Elitism and Distance Strategy for Selection of Evolutionary Algorithms. https://www.researchgate.net/publication/326852699_Elitism_and_Distance_Strategy_for_Selection_of_Evolutionary_Algorithms [data dostępu: 19.11.2024 , data publikacji: 06.08.2018]

[3] Hery Purnomo, Mauridhi & Ariyadi, Ariyadi & Nugroho, Supeno. (2014). Creep Offenses Evolution in Tower Defense Games using NSGA-II. https://www.researchgate.net/publication/308330119_Creep_Offenses_Evolution_in_Tower_Defense_Games_using_NSGA-II [data dostępu: 25.11.2024 , data publikacji: 20.09.2016]

[4] Ivan Bratko. (1986). Prolog programming for artificial intelligence. Publisher: Addison-Wesley, Wokingham, England, s. 265-285

[5] Kerssemakers, Manuel & Tuxen, Jeppe & Togelius, Julian & Yannakakis, Georgios. (2012). A procedural procedural level generator generator. 2012 IEEE Conference on Computational Intelligence and Games, CIG 2012. s. 335-341. https://www.researchgate.net/publication/261280555_A_procedural_procedural_level_generator_generator [data dostępu: 25.11.2024, data publikacji: 01.09.2012]

[6] Niu, Ben & Wang, Haibo & Ng, Peter & Shiu, Simon. (2009). A Neural-Evolutionary Model for Case-Based Planning in Real Time Strategy Games. s. 291-300, https://www.researchgate.net/publication/221048785_A_Neural-Evolutionary_Model_for_Case-Based_Planning_in_Real_Time_Strategy_Games [data dostępu: 23.11.2024, data publikacji: 24.06.2009]

[7] Unity - Engine Documentation https://www.docs.unity.com [data dostępu: 20.09.2024]