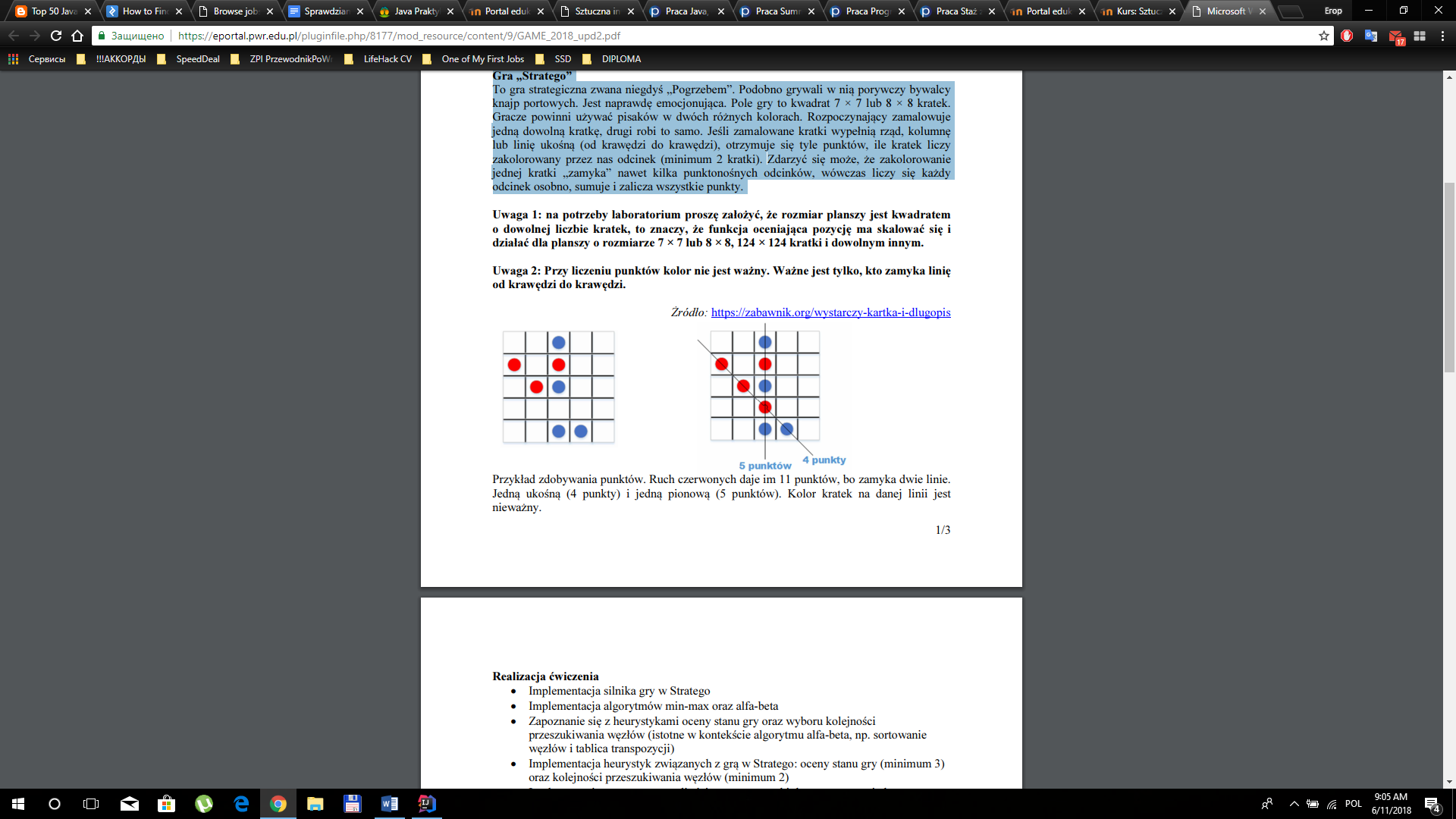
Sprawozdanie 3

Sztuczna inteligencja i inżynieria wiedzy

Yahor Yeliseyenka

1. **Wstęp teoretyczny**

Gra „Stratego” To gra strategiczna zwana niegdyś „Pogrzebem”. Podobno grywali w nią porywczy bywalcy knajp portowych. Jest naprawdę emocjonująca. Pole gry to kwadrat 7 × 7 lub 8 × 8 kratek. Gracze powinni używać pisaków w dwóch różnych kolorach. Rozpoczynający zamalowuje jedną dowolną kratkę, drugi robi to samo. Jeśli zamalowane kratki wypełnią rząd, kolumnę lub linię ukośną (od krawędzi do krawędzi), otrzymuje się tyle punktów, ile kratek liczy zakolorowany przez nas odcinek (minimum 2 kratki). Zdarzyć się może, że zakolorowanie jednej kratki „zamyka” nawet kilka punktonośnych odcinków, wówczas liczy się każdy odcinek osobno, sumuje i zalicza wszystkie punkty.



Przykład zdobywania punktów. Ruch czerwonych daje im 11 punktów, bo zamyka dwie linie. Jedną ukośną (4 punkty) i jedną pionową (5 punktów). Kolor kratek na danej linii jest nieważny.

**Algorytm MinMax**

Minimax (czasami minmax) – metoda minimalizowania maksymalnych możliwych strat. Alternatywnie można je traktować jako maksymalizację minimalnego zysku (maximin). Wywodzi się to z teorii gry o sumie zerowej, obejmujących oba przypadki, zarówno ten, gdzie gracze wykonują ruchy naprzemiennie, jak i ten, gdzie wykonują ruchy jednocześnie. Zostało to również rozszerzone na bardziej skomplikowane gry i ogólne podejmowanie decyzji w obecności niepewności.

Posiadając funkcję S oceniającą wartość stanu gry w dowolnym momencie(gracz min chce ten stan zminimalizować, a gracz max zmaksymalizować) obliczamy drzewo wszystkich możliwych stanów w grze do pewnej głębokości(ograniczonej zazwyczaj przez naszą moc obliczeniową). Zakładając, że rozgałęzienie drzewa stanów jest stałe i wynosi b(czyli na każdy ruch można odpowiedzieć b innymi) , a głębokość d(tyle ruchów do przodu symulujemy algorytmem minmax) to mamy stanów końcowych dla których obliczamy wartość stanu gry funkcją S. Zaczynamy przeglądanie od stanów końcowych, symulując optymalne wybory dla obu graczy tak aby na głębokości d(w liściach drzewa) była dla nich optymalna liczba S(stan gry po wykonaniu d ruchów). Tak więc gracz min zawsze wybiera ruch który prowadzi do mniejszej wartości końcowej, a gracz max - przeciwnie. Po przeprowadzeniu tej symulacji gracz, który znajduje się w korzeniu drzewa(aktualnie wykonujący ruch) ma pewność, że jego ruch jest optymalny w kontekście informacji o stanie gry z przeprowadzonej symulacji algorytmem minimax na głębokość d(tzn. maksymalizuje minimalny zysk).

Algorytm służy do wybrania optymalnego ruchu w danym momencie dlatego po ruchu przeciwnika musimy przeprowadzić symulację ponownie. Większa głębokość d symulacji prowadzi do lepszych ruchów. Optymalizacja algorytmu z odcięciami alfa-beta pozwala, w optymalnym przypadku, zmniejszyć ilość rozpatrywanych stanów do ~ , co w efekcie pozwala nam symulować ruchy prawie dwa razy głębiej.

Aby osiągnąć optymalne wyniki minimaxem ważne jest posiadanie dobrej funkcji oceny stanu gry S. Optymalnej funkcji S zazwyczaj nie znamy bo w takim wypadku gra byłą by już rozwiązana(znalibyśmy optymalną strategię jak np. w kółko i krzyżyk), dlatego stosuje się różne heurystyki, zazwyczaj wyrażone jako liniowy wielomian od parametrów stanu gry.

**Algorytm AlphaBeta**

Algorytm Alfa-Beta – algorytm przeszukujący, redukujący liczbę węzłów, które muszą być rozwiązywane w drzewach przeszukujących przez algorytm min-max. Jest to przeszukiwanie wykorzystywane w grach dwuosobowych, takich jak kółko i krzyżyk, szachy, go. Warunkiem stopu jest znalezienie przynajmniej jednego rozwiązania czyniącego obecnie badaną opcję ruchu gorszą od poprzednio zbadanych opcji. Wybranie takiej opcji ruchu nie przyniosłoby korzyści graczowi ruszającemu się, dlatego też nie ma potrzeby przeszukiwać dalej gałęzi drzewa tej opcji. Ta technika pozwala zaoszczędzić czas poszukiwania bez zmiany wyniku działania algorytmu.

Korzyść płynąca z algorytmu alfa-beta leży w fakcie, że niektóre gałęzie drzewa przeszukiwania mogą zostać odcięte. Czas przeszukiwania ograniczony zostaje do przeszukania najbardziej obiecujących poddrzew, w związku z czym możemy zejść głębiej w tym samym czasie. Tak samo jak klasyczny min-max, algorytm należy do algorytmów wykorzystujących metody podziału i ograniczeń (branch and bound). Współczynnik rozgałęzienia jest dwukrotnie mniejszy niż w metodzie min-max. Algorytm staje się wydajniejszy, gdy węzły rozwiązywane są układane w porządku optymalnym lub jemu bliskim.

Ze średnim albo stałym współczynnikiem rozgałęzienia b i głębokością d maksymalna liczba rozwiązanych węzłów (kiedy porządkowanie ruchów jest przypadkiem pesymistycznym) wynosi O(b\*b\*...\*b) = O(bd) i jest taki sam jak w przypadku min-max. Jeśli porządek wykonywania ruchów jest optymalny, czyli najlepsze ruchy są przeszukiwane jako pierwsze, liczba przeszukiwanych pozycji wyniesie O(b\*1\*b\*1\*...\*b) dla nieparzystej głębokości i odpowiednio O (b\*1\*b\*1\*...\*1), gdy głębokość będzie parzysta. W późniejszych przypadkach efektywny współczynnik rozgałęzienia jest redukowany do pierwiastka lub przeszukiwanie może odbywać się dwukrotnie głębiej[1]. b\*1\*b\*1\*... bierze się stąd, że wszystkie pierwsze ruchy gracza muszą zostać sprawdzone w celu znalezienia ruchu najlepszego, ale dla każdego kolejnego tylko najlepszy ruch gracza jest potrzebny, aby odrzucić wszystkie ruchy poza pierwszym, najlepszym ruchem – alfa-beta dba o to, że żaden inny ruch drugiego gracza nie musi być brany pod uwagę. Jeśli b=40 (szachy) i głębokość wynosi 12, współczynnik pomiędzy optymalnym i pesymistycznym przypadkiem współczynnika jest bliski 406.

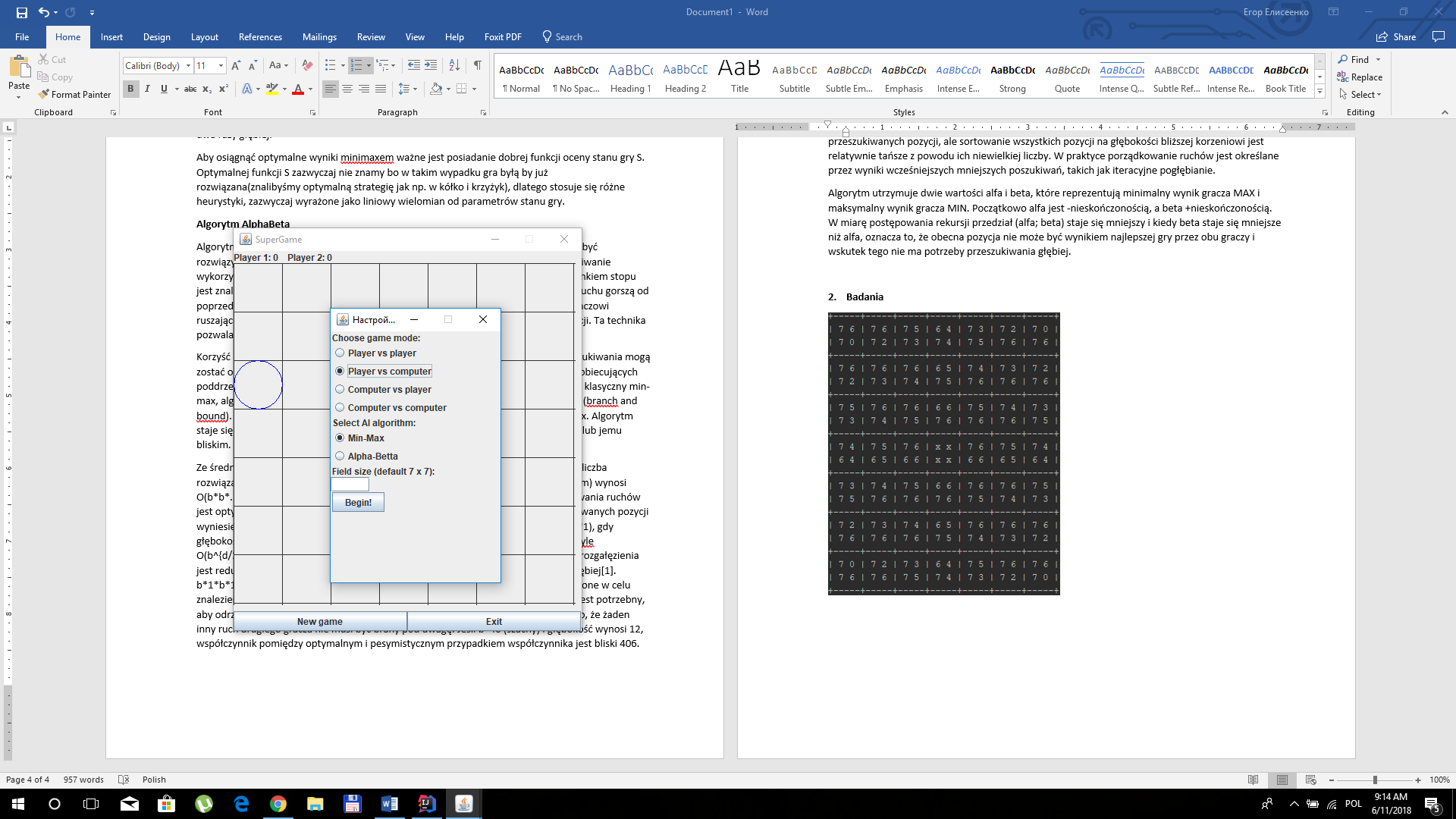
Normalnie w trakcie wykonywania algorytmu alfa-beta poddrzewa są tymczasowo zdominowane przez przewagę pierwszego gracza (kiedy ruchy gracza są dobre i w każdym wyszukiwaniu głębokość jest odpowiednia, ale każda odpowiedź drugiego gracza jest nastawiona na odparcie ataku) lub vice versa. Ta przewaga może się wiele razy powtórzyć w trakcie poszukiwań, jeśli porządek ruchów jest niewłaściwy – za każdym razem prowadząc do marnotrawstwa. Jako że liczba pozycji zmniejsza się wykładniczo dla każdego ruchu początkowego, warto zastanowić się nad sortowaniem pierwszych ruchów. Zastosowanie sortowania na każdej głębokości wykładniczo zredukuje liczbę przeszukiwanych pozycji, ale sortowanie wszystkich pozycji na głębokości bliższej korzeniowi jest relatywnie tańsze z powodu ich niewielkiej liczby. W praktyce porządkowanie ruchów jest określane przez wyniki wcześniejszych mniejszych poszukiwań, takich jak iteracyjne pogłębianie.

Algorytm utrzymuje dwie wartości alfa i beta, które reprezentują minimalny wynik gracza MAX i maksymalny wynik gracza MIN. Początkowo alfa jest -nieskończonością, a beta +nieskończonością. W miarę postępowania rekursji przedział (alfa; beta) staje się mniejszy i kiedy beta staje się mniejsze niż alfa, oznacza to, że obecna pozycja nie może być wynikiem najlepszej gry przez obu graczy i wskutek tego nie ma potrzeby przeszukiwania głębiej.

1. **Badania**

GUI.

Przy kliknięciu na przycisk nowa gra można wybrać jeden z 4ech trybów gry: dwa użytkownika, użytkownik przeciw komputerowi, komputer przeciwko użytkownikowi, oraz komputer przeciw komputerowi.

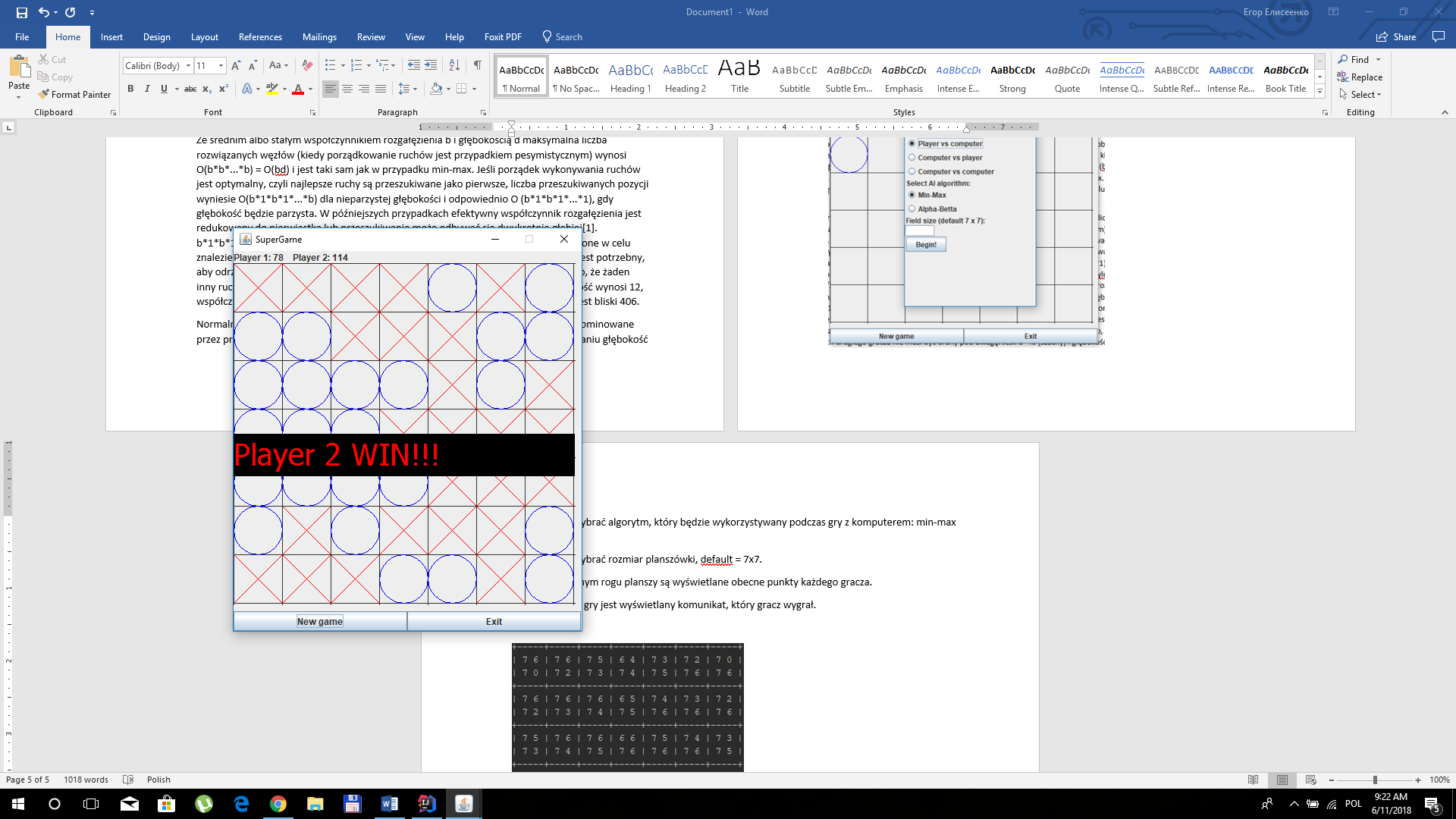


Także można wybrać algorytm, który będzie wykorzystywany podczas gry z komputerem: min-max lub alfa-beta.

Także można wybrać rozmiar planszówki, default = 7x7.

W prawym górnym rogu planszy są wyświetlane obecne punkty każdego gracza.

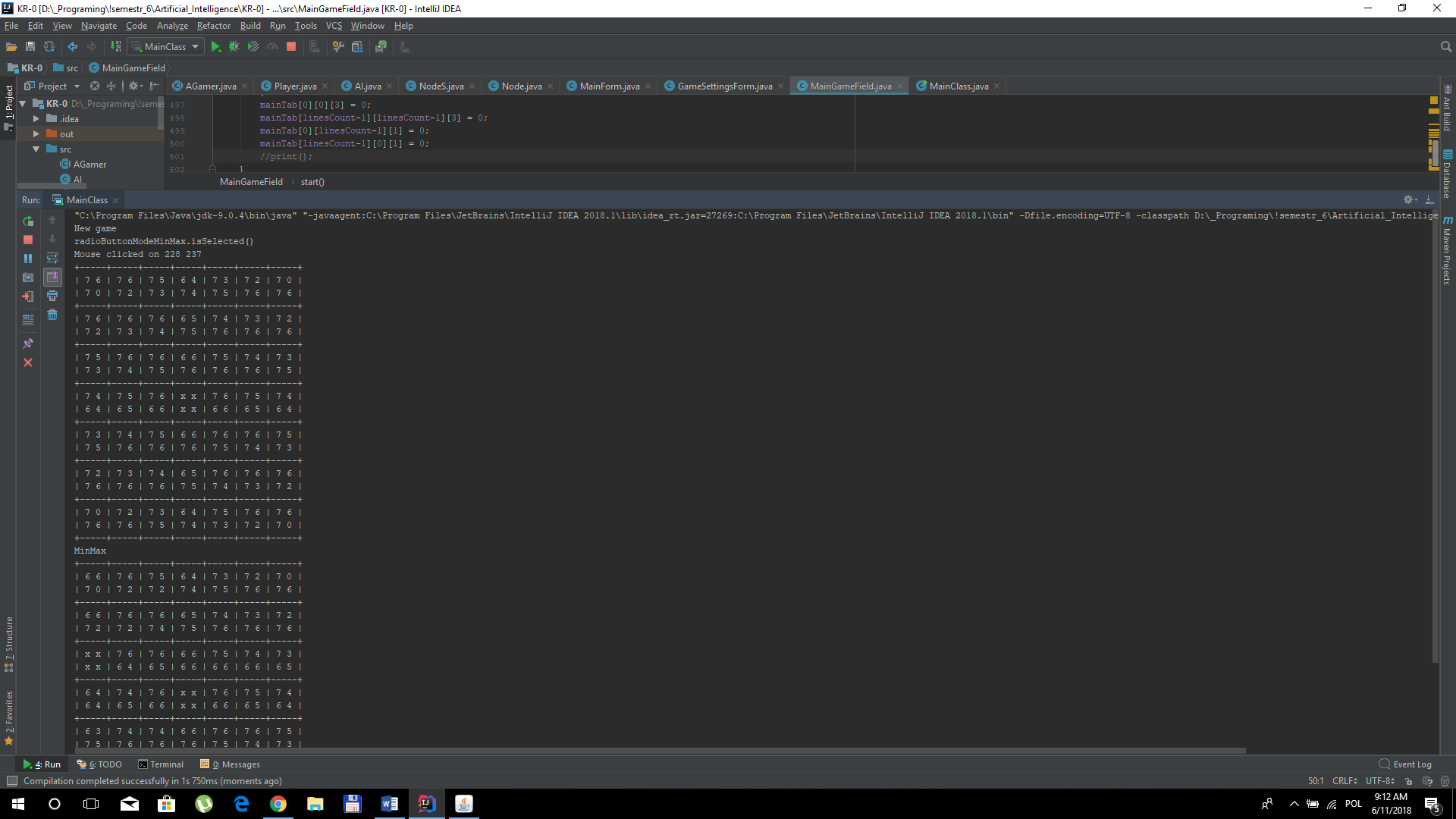
Po zakończeniu gry jest wyświetlany komunikat, który gracz wygrał.



**Heurystyki funkcji oceny.**

Działa na tablicy generowanej na początku każdej gry. Tablica jest odzwierciedlaniem planszy. Każda komórka tablicy odpowiada każdej komórce obecnej planszy. Każda komórka posiada 4y wartości. Każda wartość pokazuje jaka ilość ruchów jest potrzebna, żeby zamknąć linijkę i otrzymać punkty. Po kolei oznaczają ilość punktów: pionowo, poziomo, ukośnie od lewej do prawej, po przekątnej od prawej do lewej.

Jeżeli przynajmniej jedna z wartości jest równa 1, to f.o. oblicza punkty. W przeciwnym przypadku zwraca 0.



Tak wygląda tablica f.o. po pierwszym ruchu.

**Heurystyki wyboru kolejnego ruchu.**

1. Komórki są przeszukiwane w odpowiedniej kolejności. Dla planszy rozmiarem 7x7: 1, 3, 4, 2. To oznacza że szuka takich komórek, w których jest przynajmniej jedna z powyżej wymienionych wartości. W przypadku wartości 1, 3, 4 szuka komórek, które posiadają te wartości oraz nie posiadają 2.
2. Dla algorytmu alfa-beta są takie same wartości, jak i w pierwszej heurystyce, ale najpierw są brane pod uwagę te komórki, które mają większą liczebność powyżej wymienionych wartości.

**Porównywanie czasów działania algorytmów MinMax oraz AlfaBeta podczas 1 ruchu na planszy 7x7.**

|  |  |  |
| --- | --- | --- |
| Głębokość przeszukiwania | MinMax | AlfaBeta |
| 1 | 120598 s^-9 | 130006 s^-9 |
| 2 | 690659 s^-9 | 197576 s^-9 |
| 3 | 3 s^-3 | 1 s^-3 |
| 4 | 6 s^-3 | 1 s^-3 |
| 5 | 39 s^-3 | 5 s^-3 |
| 6 | 166 s^-3 | 18 s^-3 |
| 7 | 1996 s^-3 | 35 s^-3 |

Jak widać z wykresu wyżej, algorytm AlfaBeta działa szybciej.

**Porównywanie ilości przeszukanych węzłów algorytmów MinMax oraz AlfaBeta podczas 1 ruchu na planszy 7x7.**

|  |  |  |
| --- | --- | --- |
| Głębokość przeszukiwania | MinMax | AlfaBeta |
| 1 | 12 | 9 |
| 2 | 120 | 23 |
| 3 | 768 | 79 |
| 4 | 2712 | 1131 |
| 5 | 41592 | 2561 |
| 6 | 718104 | 35527 |
| 7 | 10702488 | 72002 |

Jak widać z wykresu wyżej, algorytm AlfaBeta przeszukuje mniej węzłów.

**Skuteczność.**

Skuteczność wykonanych ruchów oceniłem przez kilka rozgrywek. W których:

- głębokość przeszukiwania MinMax = 5, AlfaBeta = 5, pierwszy ruch wykonuje MinMax, Rozmiar planszy 7x7

- głębokość przeszukiwania MinMax = 5, AlfaBeta = 5, pierwszy ruch wykonuje MinMax, Rozmiar planszy 8x8

- głębokość przeszukiwania MinMax = 5, AlfaBeta = 5, pierwszy ruch wykonuje AlfaBeta, Rozmiar planszy 7x7

- głębokość przeszukiwania MinMax = 5, AlfaBeta = 5, pierwszy ruch wykonuje AlfaBeta, Rozmiar planszy 8x8

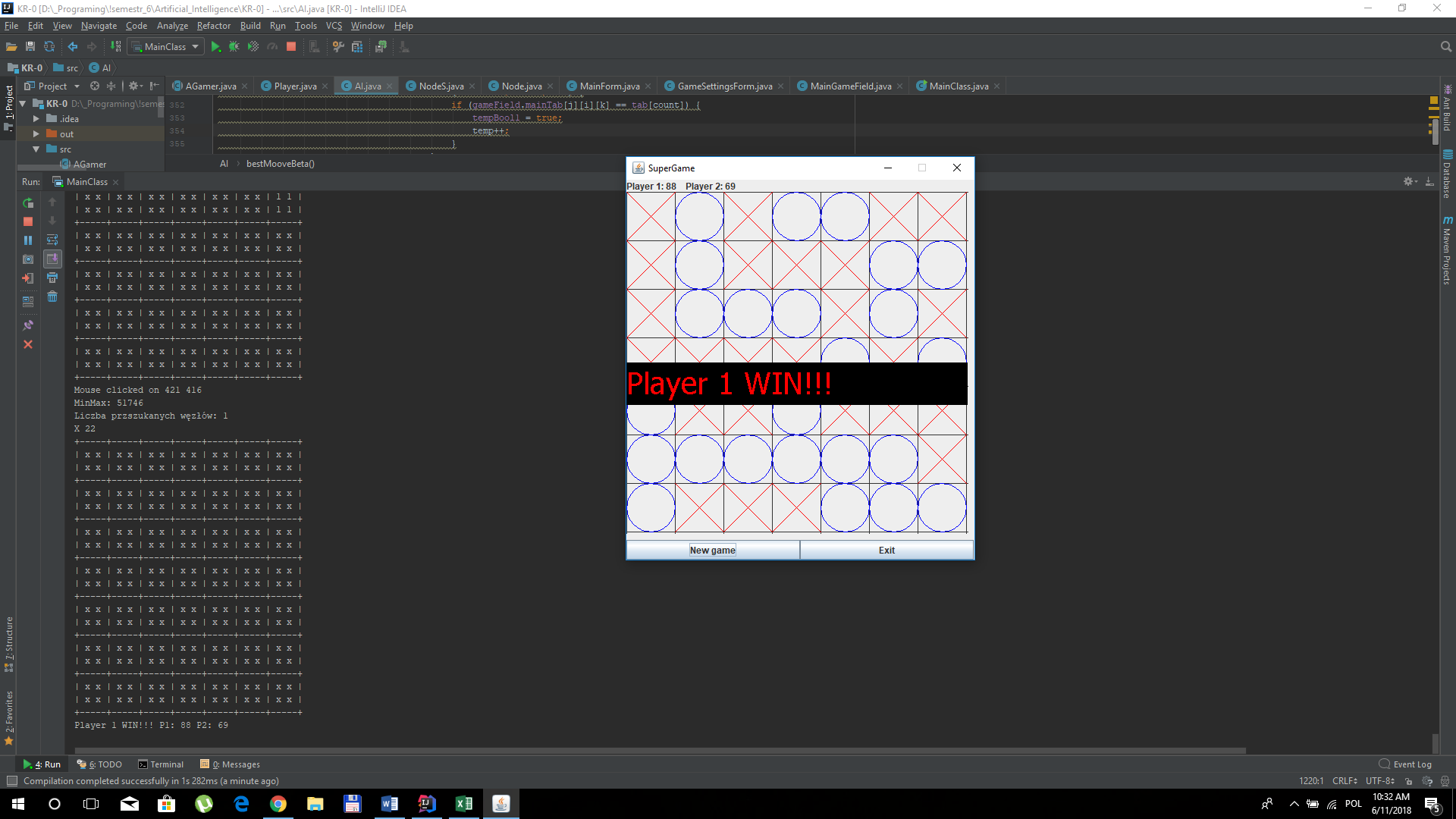
- głębokość przeszukiwania MinMax = 6, AlfaBeta = 6, pierwszy ruch wykonuje MinMax, Rozmiar planszy 7x7

- głębokość przeszukiwania MinMax = 6, AlfaBeta = 6, pierwszy ruch wykonuje MinMax, Rozmiar planszy 8x8

- głębokość przeszukiwania MinMax = 6, AlfaBeta = 6, pierwszy ruch wykonuje AlfaBeta, Rozmiar planszy 7x7

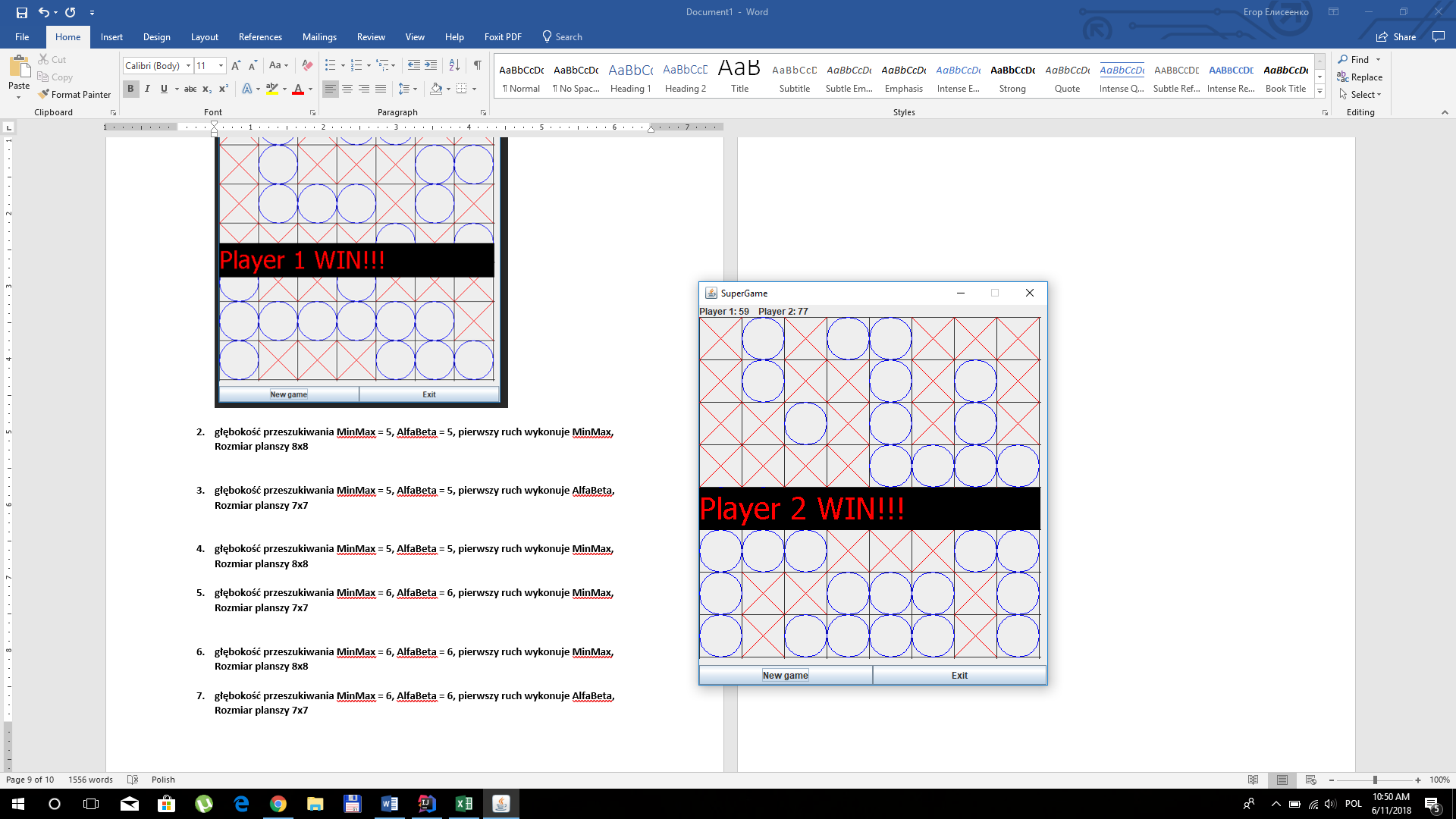
- głębokość przeszukiwania MinMax = 6, AlfaBeta = 6, pierwszy ruch wykonuje AlfaBeta, Rozmiar planszy 8x8

1. **Głębokość przeszukiwania MinMax = 5, AlfaBeta = 5, pierwszy ruch wykonuje MinMax, Rozmiar planszy 7x7**



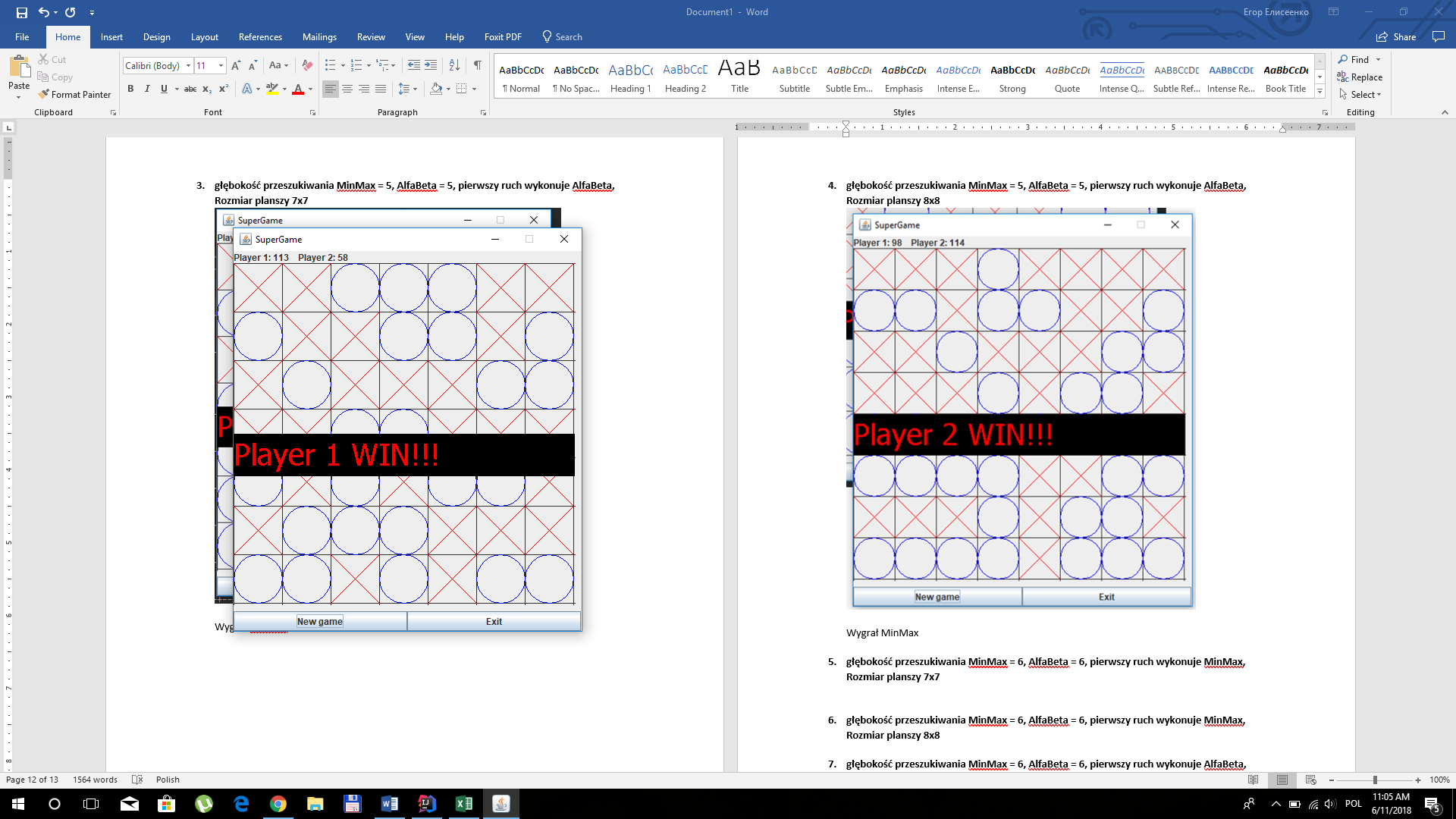
Wygrał MinMax

1. **głębokość przeszukiwania MinMax = 5, AlfaBeta = 5, pierwszy ruch wykonuje MinMax, Rozmiar planszy 8x8**



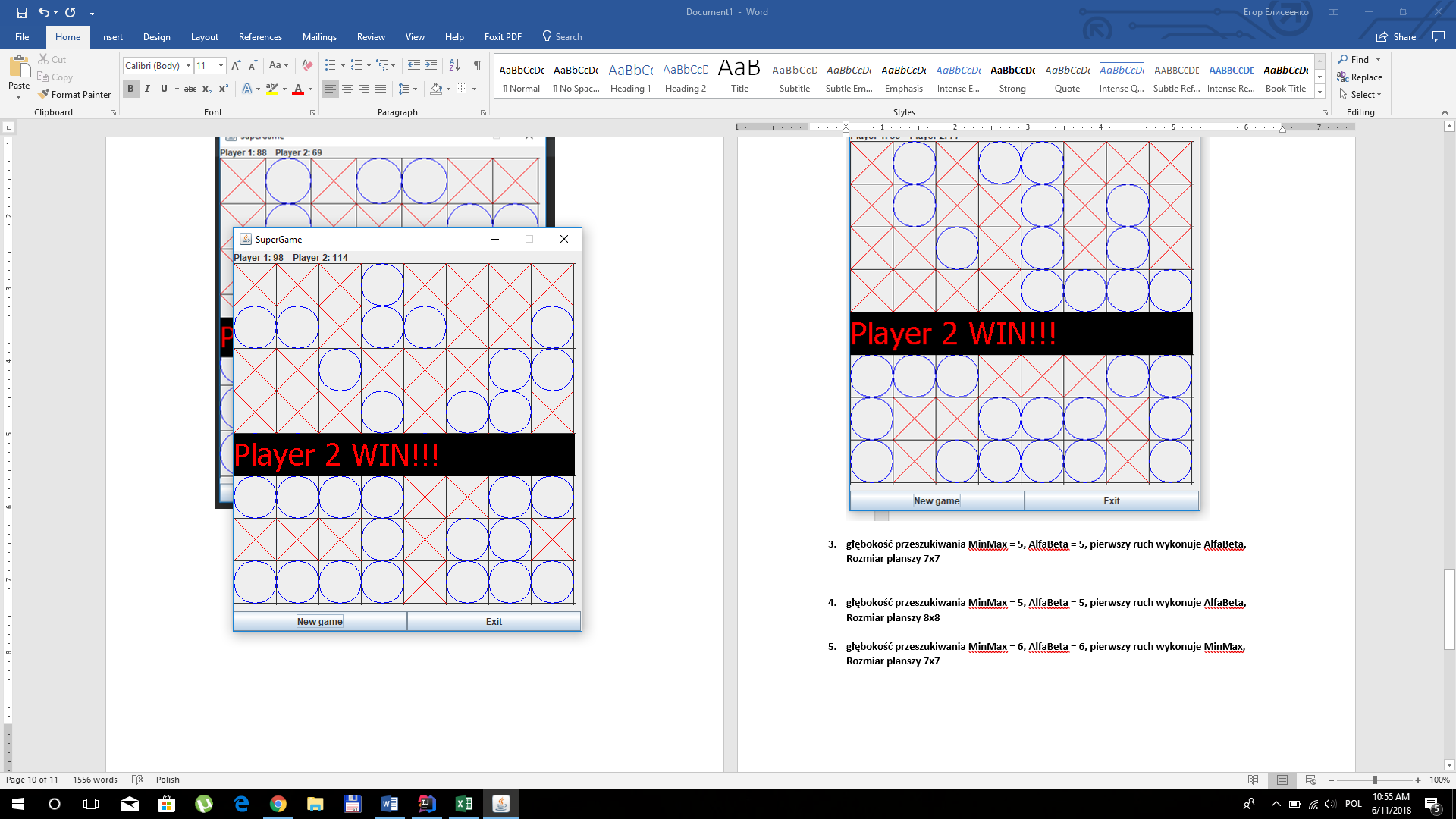
Wygrał AlfaBeta

1. **głębokość przeszukiwania MinMax = 5, AlfaBeta = 5, pierwszy ruch wykonuje AlfaBeta, Rozmiar planszy 7x7**



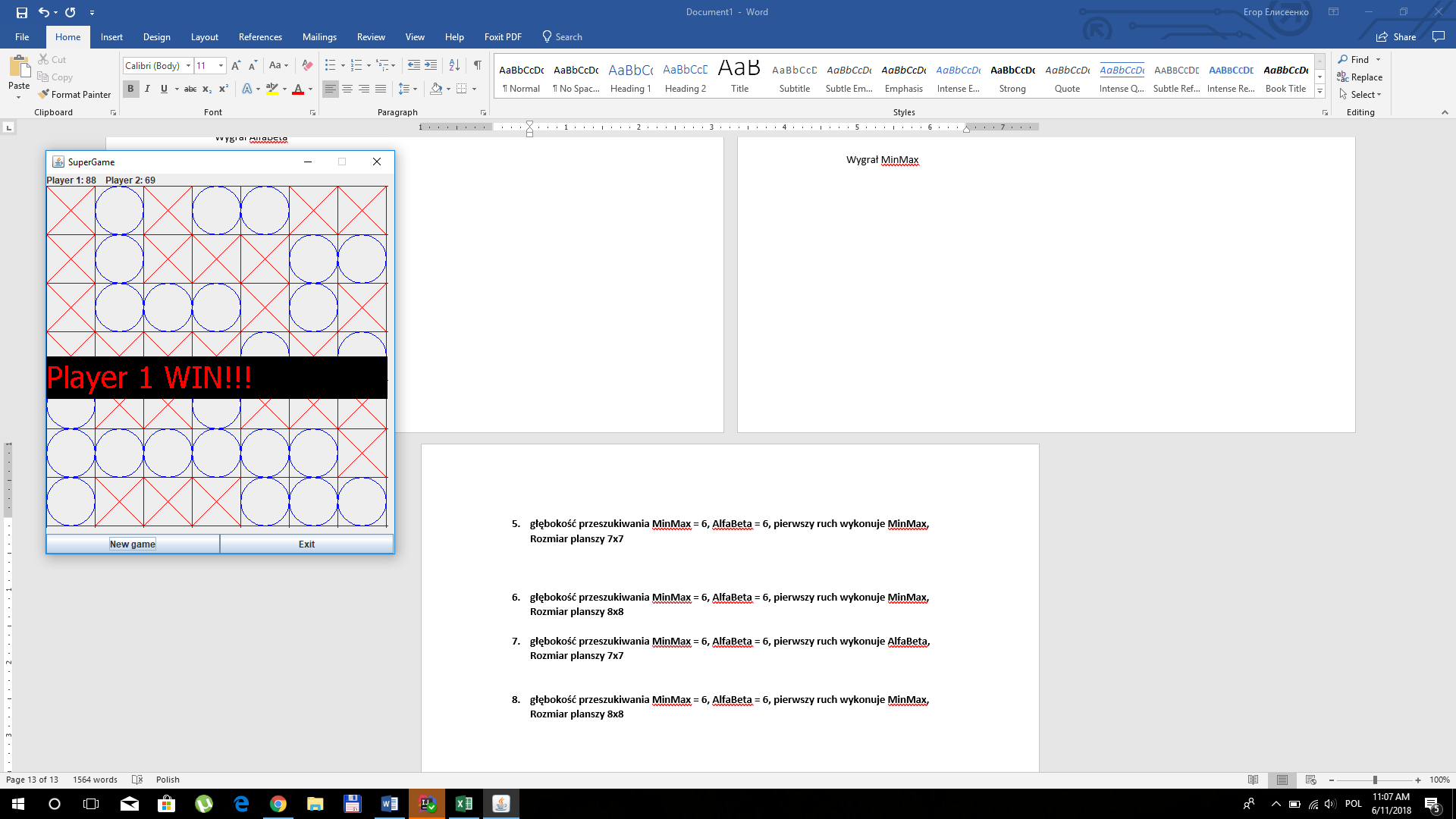
Wygrał AlfaBeta

1. **głębokość przeszukiwania MinMax = 5, AlfaBeta = 5, pierwszy ruch wykonuje AlfaBeta, Rozmiar planszy 8x8**



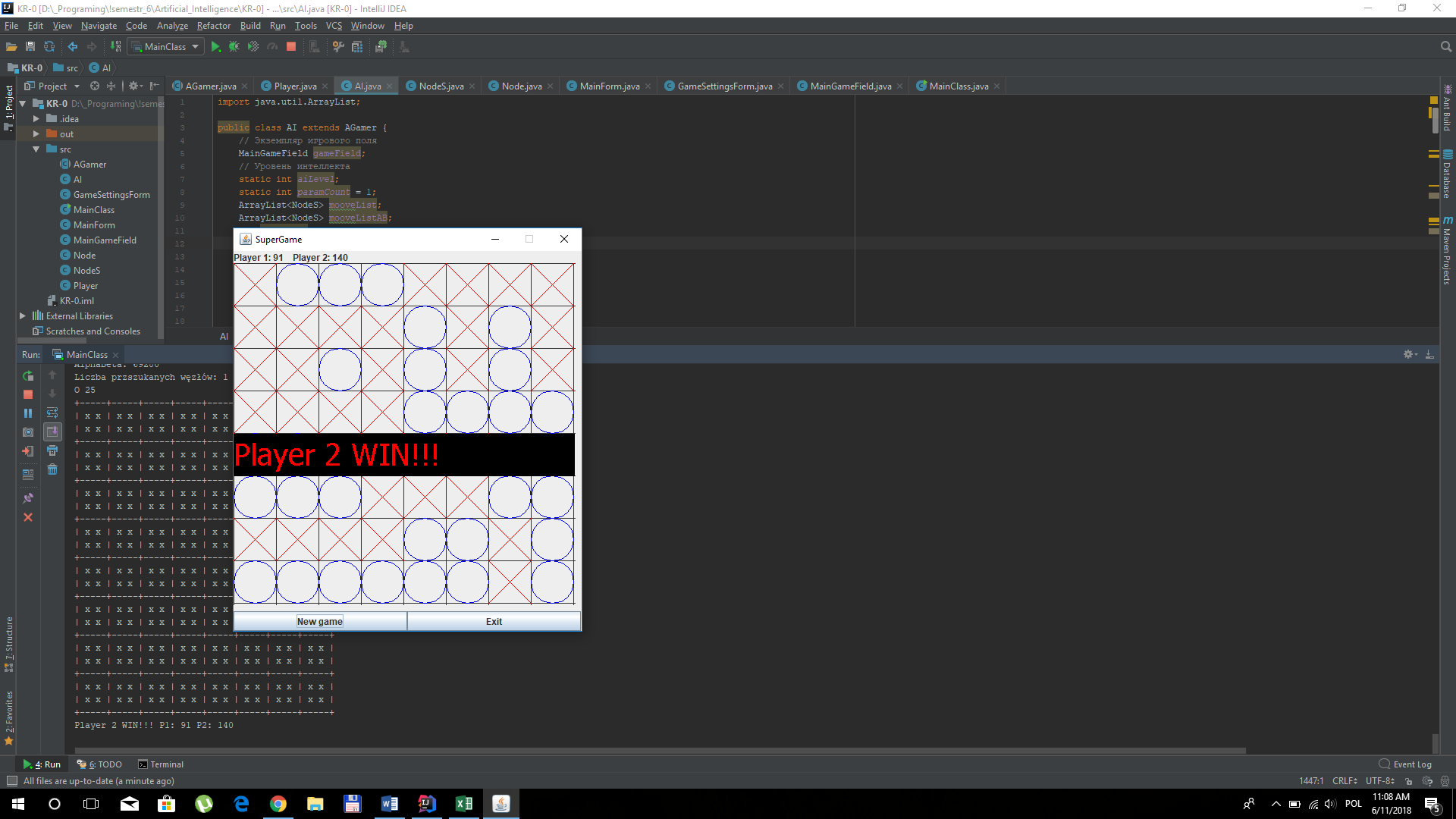
Wygrał MinMax

1. **głębokość przeszukiwania MinMax = 6, AlfaBeta = 6, pierwszy ruch wykonuje MinMax, Rozmiar planszy 7x7**



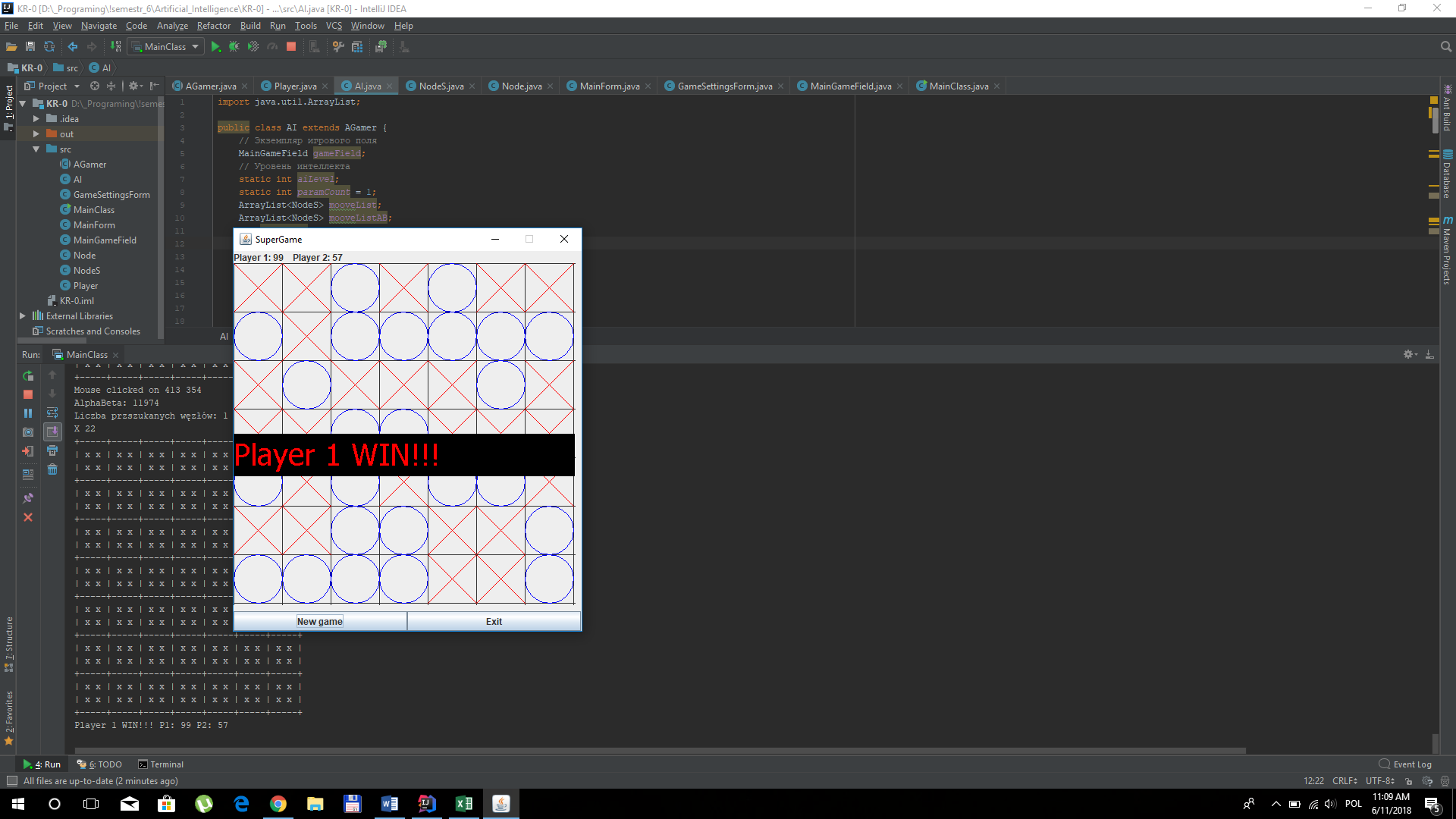
Wygrał MinMax

1. **głębokość przeszukiwania MinMax = 6, AlfaBeta = 6, pierwszy ruch wykonuje MinMax, Rozmiar planszy 8x8**



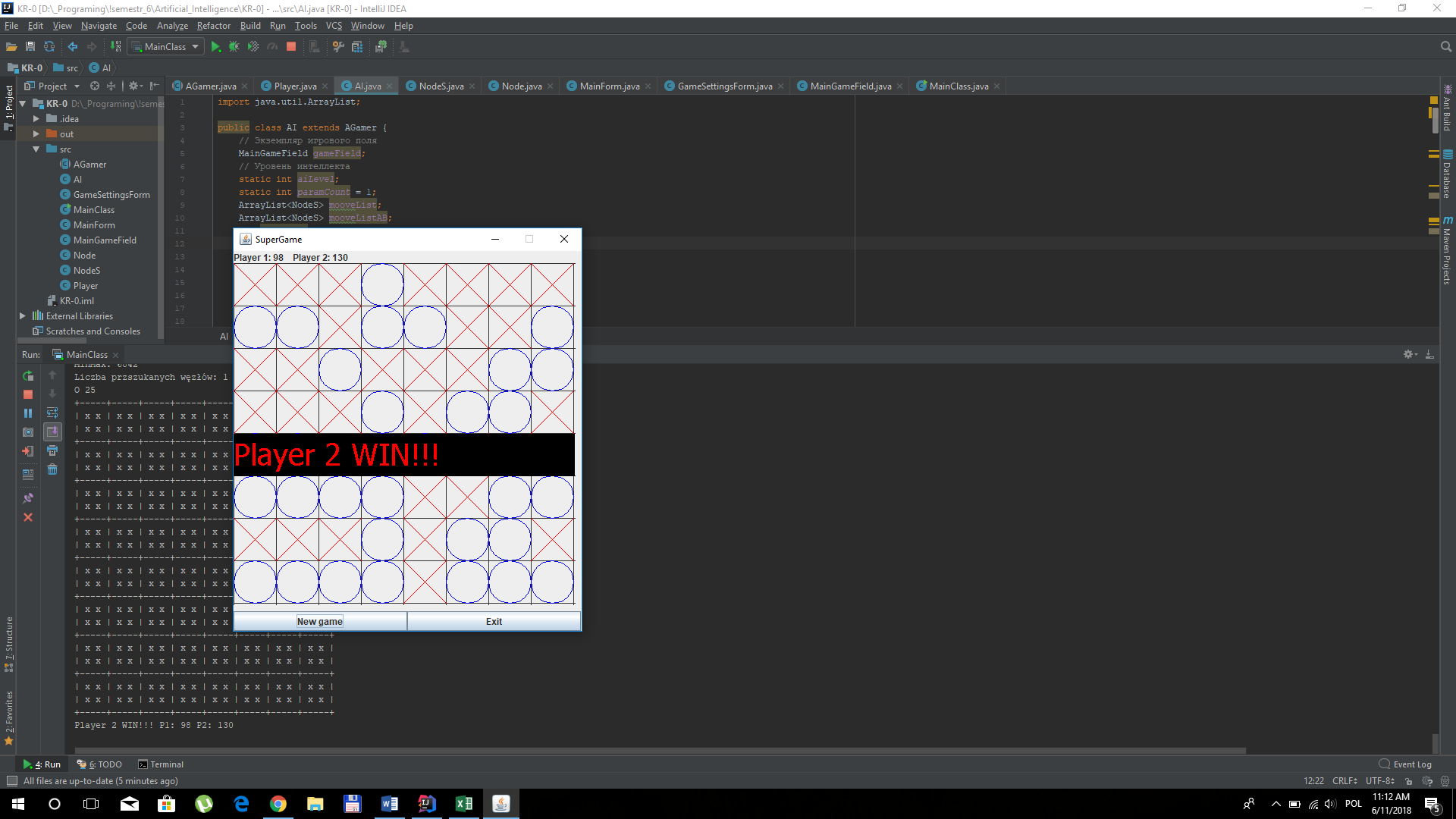
Wygrał AlfaBeta

1. **głębokość przeszukiwania MinMax = 6, AlfaBeta = 6, pierwszy ruch wykonuje AlfaBeta, Rozmiar planszy 7x7**



Wygrał AlfaBeta

1. **głębokość przeszukiwania MinMax = 6, AlfaBeta = 6, pierwszy ruch wykonuje AlfaBeta, Rozmiar planszy 8x8**



Wygrał MinMax

Ze wszystkich rozgrywek można zrobić wnioski, że zaimplementowane algorytmy działają z tą samą skutecznością. Liczba wygranych MinMax = 4 oraz liczba wygranych AlfaBeta = 4. Z tego co widać zawsze wygrywa ten algorytm, który robi ostatni ruch.

Najlepsze wyniki, czyli sytuacje w których algorytmy radzą sobie lepiej oraz nabrali więcej punktów, niż przeciwniki są następne:

AlfaBeta

1. 7x7 głębokość przeszukiwania 6 pierwszy AlfaBeta
2. 7x7 głębokość przeszukiwania 5 pierwszy AlfaBeta
3. 8x8 głębokość przeszukiwania 6 pierwszy MinMax

MinMax

1. 8x8 głębokość przeszukiwania 6 pierwszy AlfaBeta
2. 8x8 głębokość przeszukiwania 5 pierwszy AlfaBeta

Z tego można zrobić wnioski, że algorytm AlfaBeta troszeczkę lepiej sobie radzi.

**3.Wnioski**

Skuteczność algorytmów AlfaBeta oraz MinMax jest ta sama. Zawsze wygrywa ten algorytm, który robi ostatni ruch. AlfaBeta – potrzebuje mniej czasu na przeszukiwanie węzłów dla tej samej głębokości co MinMax. Ale przeszukuje mniej węzłów, co może być powodem mniejszej skuteczności.