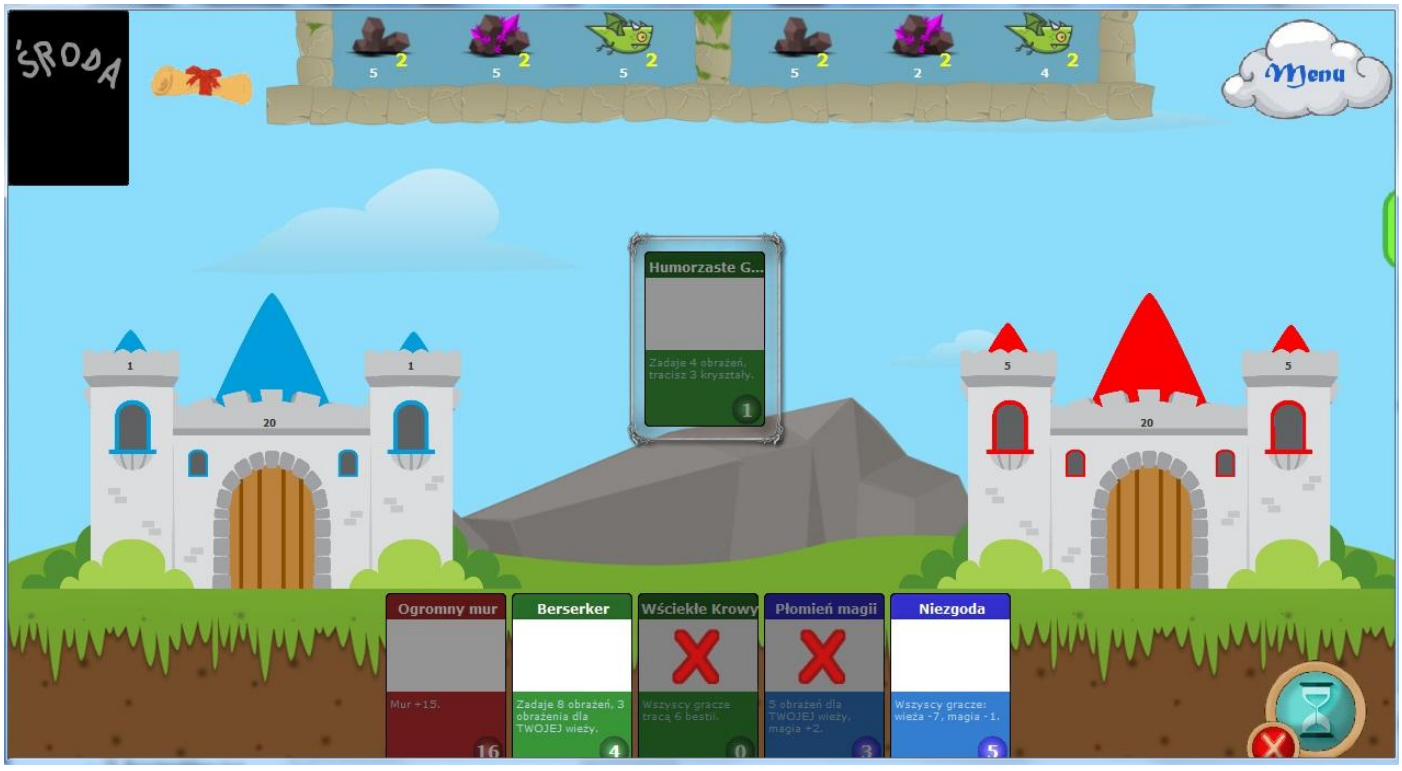
Bartłomiej Seliga, Mateusz Jaworski 167433

Dokumentacja gry karcianej "Tutaj ma byc nazwa"

1.Opis wymagań

- rozgrywkę w grę "***Tutaj ma byc nazwa***" gracza będącego człowiekiem z komputerem według podanych niżej zasad
- zapisanie i wczytanie gry
- wybranie rozdzielczości
- możliwość rozgrywki na pełnym ekranie
- wybranie poziomu trudności gry
- zapamiętywanie wybranych ustawień



Zasady gry:

Każdy z graczy rozpoczyna mając pięć kart w ręce. Gra polega na naprzemiennym zagrywaniu kart przez graczy w swoich turach. Podczas swojej tury gracz może zagrać jedną kartę (niektóre karty pozwalają na zagranie kolejnej), przeciągając ją na środek ekranu w widoczny prostokątny kontur, opuścić turę nie zagrywając żadnej karty (przycisk klepsydry) oraz wymianę niechcianych kart kosztem tury (zaznaczenie takiej karty prawym przyciskiem myszy i kliknięcie przycisku X). Na koniec każdej tury karty graczy są uzupełniane do pięciu kart w ręce. Gracze dobierają

karty z talii, w której znajduje się 85 różnych kart. Po zagraniu karta wraca do tej talii, więc nie jest możliwa sytuacja, że skończą się karty. Na górze karty znajduje się jej nazwa, a na dole opis, czyli co się dzieje po jej zagraniu.

Każdy z graczy zaczyna z taką samą wysokością wieży i muru oraz tą samą liczbą zasobów i ich przyrostem (liczba zasobów, które otrzymujemy co turę). Wartości te zależą od poziomu trudności. Zasoby są widoczne na górze ekranu gry - po lewej gracza człowieka, a po prawej komputera. Duże liczby oznaczają przyrost, a małe liczbę zasobów. Każda karta wymaga określonej ilości zasobów. Jeśli gracz nie będzie miał ich wystarczająco dużo, nie będzie mógł zagrać danej karty (będzie ona wtedy przyciemniona). Koszt karty znajduje się w jej prawym dolnym rogu. Zasoby dzielą się na trzy rodzaje (od lewej):

-kamienie - wydobywane przez *kopalnie* (taką nazwę nosi przyrost kamieni) są wymagane do rzucania czerwonych kart związanych z obroną
-kryształy - produkowane przez *magię*, umożliwiają rzucanie niebieskich kart związanych z wieżą
-bestie - hodowane w *podziemiach*, są konieczne do zagrywania zielonych kart związanych z atakiem.
Wartości zasobów oraz ich przyrostu nie mogą spaść poniżej zera.

Każdy z graczy posiada wieżę oraz mur (po lewej stronie człowiek niebieskie, po prawej komputer czerwone). Gdy wysokość wieży gracza spadnie do zera, to wtedy gracz ten przegrywa. Mur natomiast ochrania wieżę gracza. Jeśli przeciwnik zagra kartę zadającą obrażenia, to wtedy są one w pierwszej kolejności zadawane murowi, chyba że opis mówi, że obrażenia odnosi bezpośrednio wieża. Wysokości nie mogą spaść poniżej zera. O tym jak wysokie są wieża oraz mur informują znajdujące się na nich liczby, ale także odpowiednia grafika, która zmienia się przy zmianie wysokości wieży oraz muru.

Gra kończy się gdy:
-któryś z graczy rozbuduje wieżę do określonego poziomu (zależy od poziomu trudności)
-któryś z graczy zgromadzi odpowiednią ilość surowca (ilość zależy od poziomu trudności)
-wysokość wieży któregoś z graczy wyniesie zero.
Z prawej strony znajduje się zielony przycisk, po najechaniu którego pojawia się informacja o tym ile brakuje nam wysokości wieży oraz liczby zasobów do zwycięstwa.

Po zakończeniu rozgrywki pokazuje się ekran informujący gracza człowieka o wygranej lub przegranej.

2.Opis najważniejszych klas

Klasa **Gra** jest klasa dziedziczącą po JFrame, ma ona za zadanie wyświetlanie głównego okna aplikacji i zarządzanie aktualnie wyświetlanym ekranem. Umożliwia zmianę rozdzielczości, włączenie i wyłączenie pełnego ekranu.

Abstrakcyjna klasa **Ekran** wraz z podklasami w konstruktorze przyjmują referencję na JFrame i na nim się wyświetlają. Wszystko co wyświetla się na ekranie komputera jest dodane do konkretnej podklasy klasy Ekran. Każda z podklas ma metodę makeGUI która tworzy i ustawia wszystkie widoczne komponenty. Implementują również interfejs ActionListener do reagowania na kliknięcie myszą.

Klasa **Menu** dziedzicząca po klasie Ekran zawiera przyciski pozwalające na rozpoczęcie nowej gry, wczytanie już istniejącego zapisu, zmianę ustawień oraz wyjście z gry.

W klasie **Granie** dziedziczącej po klasie Ekran odbywa się rozgrywka. Są tam widoczne karty gracza będącego człowiekiem, stos, z którego dobierane są karty, stos ostatnio rzuconych kart, stan zasobów oraz zamków obu graczy. Oprócz tego znajdują się tam przyciski, które umożliwiają zapis gry, wyjście do menu, pominięcie tury, wymianę kart oraz przycisk informujący o warunkach wygranej.

Karta - Klasa ta wyświetla kartę na ekranie, pozwala również na użycie karty. W konstruktorze tworzona jest odpowiednia karta, ustawiana jest nazwa karty, jej opis oraz grafika. Karta może mieć trzy stany:

- zakryta: wszystkie dane są zakryte, wyświetlany jest jedynie rewers karty.
- odkryta: stan normalny.
- wyrzucana: tak karte można oznaczyć tylko jeżeli nalezy ona do gracza, pojawia się na niej czerwony X, tak oznaczoną kartę można wyrzucić z talii.

Metody wczytajUmiejętności(int []) oraz boolean uzyj(Gracz, Gracz) pozwalają na ustawienie odpowiednich parametrów karty oraz wykorzystanie jej.
Klasa Karta posiada zestaw modyfikatorów, które umożliwiają zmianę różnych elementów np. nazwy lub opisu. Jest to klasa abstrakcyjna po której dziedziczą trzy podklasy: KartaMagii, KartaPodziemii, KartaKopalnii, stanowią one szczególne przypadki Karty i zależnie od typu karty tworzone są odpowiednie instancje jej podklas.

- eventKarta - klasa dziedzicząca po MouseAdapter, wykrozystywana jest do interakcji z kartami. W konstruktorze przyjmuje referencje na kartę, która będzie nasłuchiwała zdarzeń z tej klasy.
- void mousePressed(MouseEvent) - jeżeli zostanie wykryte wciśnięcie przycisku, to karta na którą wskazuje kursor myszy powiadamia, ze została podniesiona.
- void mouseDragged(MouseEvent) - wykrywa przeciaganie karty, zmienia pozycję wyświetlania karty na ekranie, w raz z poruszaniem się kursora myszy.
- void mouseReleased(MouseEvent) - wykrywa puszczenie przycisku myszy, jeżeli karta w tym momencie znajduje się nad talią, na którą może upaść, to zostaje do niej dodana, jeżeli nie to karta wraca na swoje miejsce.
- void mouseClicked(MouseEvent) - jeżeli na karcie został wciśnięty prawy przycisk myszy, zostaje zmieniony jej stan.
- void mouseEntered(MouseEvent) oraz void mouseExited(MouseEvent) - są odpowiedzialne za lekkie wysuwanie karty w górę po najechaniu na nie myszą.

Klasa **taliaKart**: służy do zarządzania Kartami. W grze istnieją 4 instancje tej klasy. Pierwsza z nich to stosZakryty w klasie Granie. stosZakryty na poczatku gry wczytuje z pliku wszystkie karty i przez całą rozgrywkę z niego są pobierane karty dla graczy. Drugą instancją jest stosOdkryty również w klasie Granie. stosOdkryty przechowuje karty rzucane przez gracz, co turę jest czyszczony, a jego karty wracają na stosZakryty. Dwie kolejne instancje to talie Graczy, czyli karty które trzymają w ręku.
Najważniejsze prywatne pola to:

- talia - lista kart należących do talii;
- location - miejsce wyświetlania kart;
- friends - talie, z którymi może wchodzić w interakcje (np. przenosić karty)
- gracz - jeżeli talia nalezy do gracza, to jest to referencja na niego;
- layout - umożliwia sterowanie wyświetlaniem się kart (jedna na drugiej, jedna obok drugiej, jedna nad drugą)
- klasa ta w konstruktorze przyjmuje trzy argumenty, dwa pierwsze to pozycja na ekranie trzeci pozwala ustawić czy karty dodane do talii mają być odwrócone czy zakryte. Do najważniejszych metod należą:
- void dodaj(Karta) - pozwala dodać kartę do talii, metoda ta jest przeciążona umożliwia również zdecydowanie czy dodawana karta ma być odkryta czy zakryta po dodaniu (domyślnie jest tak jak ustawimy w konstruktorze);
- Karta wez() - pozwala na zabranie z talii karty, Karta wez(Karta) przeladowana wersja ten funkcji pozwala zabrać kartę podaną jako argument, oraz Karta wez(int) pozwala na zabranie karty o odpowiednim indeksie. Funkcje te zwracają zabieraną kartę;
- boolean kolizja(Karta) - metoda ta zwraca true gdy karta podana jako argument jest w obrębie danej talii, w przeciwnym razie zwraca false;
- private void rozlizKart() - rozkłada karty według ustalonego wewnętrzego layoutu;
- boolean toFriends(Karta) - sprawdza czy któraś z kart należących do talii wchodzi w kolizję z inną talią, jeżeli tak jest traktowana jako rzucona, zostają użyte umiejętnosci karty na graczu oraz jego przeciwniku;
- static void WczytajZPliku(String, taliaKart) - statyczna metoda pozwalająca na wczytanie kart z pliku tekstowego. Pierwszym argumentem jest ścieżka do pliku, drugim talia do której mają się wczytać karty.

Klasa **Ustawienia** implementuje wzorzec projektowy Singleton, dzięki czemu możliwy jest tylko jeden obiekt tej klasy oraz jest do niej globalny dostęp. Odpowiada za wczytywanie i zapisywanie ustawień programu i za zmianę poziomu trudności gry. Ustawienia są zapisywane do pliku i wczytywane przy uruchamianiu programu.

Klasa **Gracz**, dodaje nowego gracza sterowanego przez człowieka (podklasa Człowiek) lub przez komputer (podklasa NPC). Ustawia odpowiednie komponenty na ekranie, wczytuje z ustawień początkowe dane gracza. Tworzy odpowiednią instancję klasy taliaKart, w której są trzymane karty do wykorzystania przez gracza ludzkiego lub komputerowego. Ma odpowiednie metody powiadamiające o przegranej/wygranej gracza.

Klasa **StanGry** służy do zapisywania i wczytywania gry. Cały proces odbywa się za pomocą serializacji i deserializacji. Klasa ta implementuje interfejs Serializable.

Klasa **EventKursor** dziedzicząca po klasie MouseAdapter odpowiada za zmianę wyglądu kursora podczas klikania, trzymania oraz puszczania klawiszy myszki.

Klasa **Przycisk** dziedziczy po klasie JButton. Klasa ta jest wykorzystywana do tworzenia wszystkich przycisków znajdujących się w grze. Ściśle związana z nią jest klasa EventPrzycisk, która odpowiada za zmianę wyglądu przycisku po najechaniu na niego myszką oraz po jego opuszczeniu.

Obrazek - klasa zarządzajaca grafiką w grze. W konstruktorze otrzymuje poprzez parametry ścieżkę do grafiki, miejsce wyświetlania oraz rozmiary grafiki. Klasa wykorzystuje wewnętrzną HashMape. Kluczem HaszMapy jest String a wartością BufferedImage. Klasa najpierw przeszukuje HashMap'ę aby sprawdzić czy nie ma w niej już grafiki pasującej do aktualnie wczytywanej. Jeżeli klucz który pasuje do ścieżki aktualnie wczytywanej grafiki to ustawia grafikę jako wartość z HashMapy, jeżeli nie znajdzie odpowiedniego klucza, grafika jest pobierana z dysku twardego, a następnie zapisywana do HasMapy.
Klasa ta ma statyczną metodę resize która pozwala na zmianę rozmiaru grafiki pobranej jako argument.

3.Opis realizacji funkcjonalności

- Rozgrywka została zrealizowana za pomocą GUI. Do przedstawienia GUI wykorzystaliśmy biblioteki swing odpowiednio dostosowując komponenty takie jak JLabel czy JButton. Gracz poprzez naciskanie przycisków myszy lub przeciąganie elementów po ekranie ma możliwość wchodzenia w interakcję z programem. Podzieliliśmy grę na odpowiednie ekrany, które ułatwiają zarządzanie wyświetlanymi elementami oraz poprawiają intuicyjność gry.
- Zapisywanie oraz wczytywanie gry zostało zrealizowane za pomocą serializacji niezbędnych danych oraz ich odtworzeniu przy próbie wczytania gry. Służy do tego klasa StanGry.
- Wybrane przez użytkownika ustawienia zostają zapamiętane w pliku tekstowym w momencie opuszczenia ekranu ustawień i są odtwarzane przy następnym uruchomieniu gry.
- Zmiana rozdzielczości została zrealizowana poprzez podanie nowych wartości szerokości i wysokości okna, które są przekazywane do obiektu Ustawienia. Następnie wyliczana jest nowa skala każdego obiektu : (rozdzielczoscX / NATYWNA_ROZDZIELCZOSC_X; gdzie rozdzielczoscX to rozdzielczosc, na którą chcemy zmienić, a NATYWNA_ROZDZIELCZOSC_X to stała wynosząca 1600). Po wyliczeniu skali następuje usunięcie wszystkich widocznych ekranów, a następnie dodanie ich popraz kolejny po przeskalowaniu. Również okno gry zmienia swój rozmiar do podanych na początku wartości. Jeżeli gra jest uruchomiona w trybie pełnego ekranu, okno dopasuje się do ekranu na którym jest wyświetlane. Natomiast jeżeli wyświetlamy grę w wyłączonym trybie pełnego ekranu, to jest możliwość wybrania jednej z trzech możliwości: 1280x720, 1600x900 i 1920x1080. Są to wartości stałe i nie można ich zmieniać. Wybór jednej z rozdzielczości można dokonać w ustawieniach gry, pod warunkiem wyłączenia trybu pełnego ekranu.
- Poziom trudności wpływa na wartości zmiennych takich jak: WYGRANA_ZASOBY, WYGRANA_WIEZA, POCZATKOWA_WIEZA, POCZATKOWY_MUR, POCZATKOWE_KOPALNIE,POCZATKOWE_ZASOBY; Przy zmianie poziomu trudności w ustawieniach gry, zmienne te są podwyższane/obniżane aby utrudnić/ułatwić grę. Funkcja zmieniająca poziom trudności przyjmuje w argumencie typ enum określający nowy poziom trudności (PoziomTrudnosci.latwy, PoziomTrudnosci.sredni, PoziomTrudnosci.trudny).

4.Trwałość danych

Stan gry jest zapisywany za pomocą serializacji. Zapisywane są elementy takie jak stany zasobów obu graczy, wysokość ich wież oraz murów, a także informacja jakie karty mają w ręku poszczególni gracze. Następnie przy wczytywaniu gry informacje te są odtwarzane z pliku. Wykorzystywana w tym procesie jest klasa StanGry implementująca interfejs Serializable. Posiada ona nadpisane metody *void writeObject(ObjectOutputStream)* oraz *void readObject(ObjectInputStrem)*. Jednocześnie może być zapisana tylko jedna gra. Oprócz tego do pliku tekstowego zapisywane są ustawienia gry, które są wczytywane przy kolejnym uruchomieniu programu. Wykorzystywane są do tego klasy PrintWriter, File, oraz Scanner. Dzięki temu nie jest konieczna zmiana ustawień przy każdym uruchomieniu gry.

5.GUI

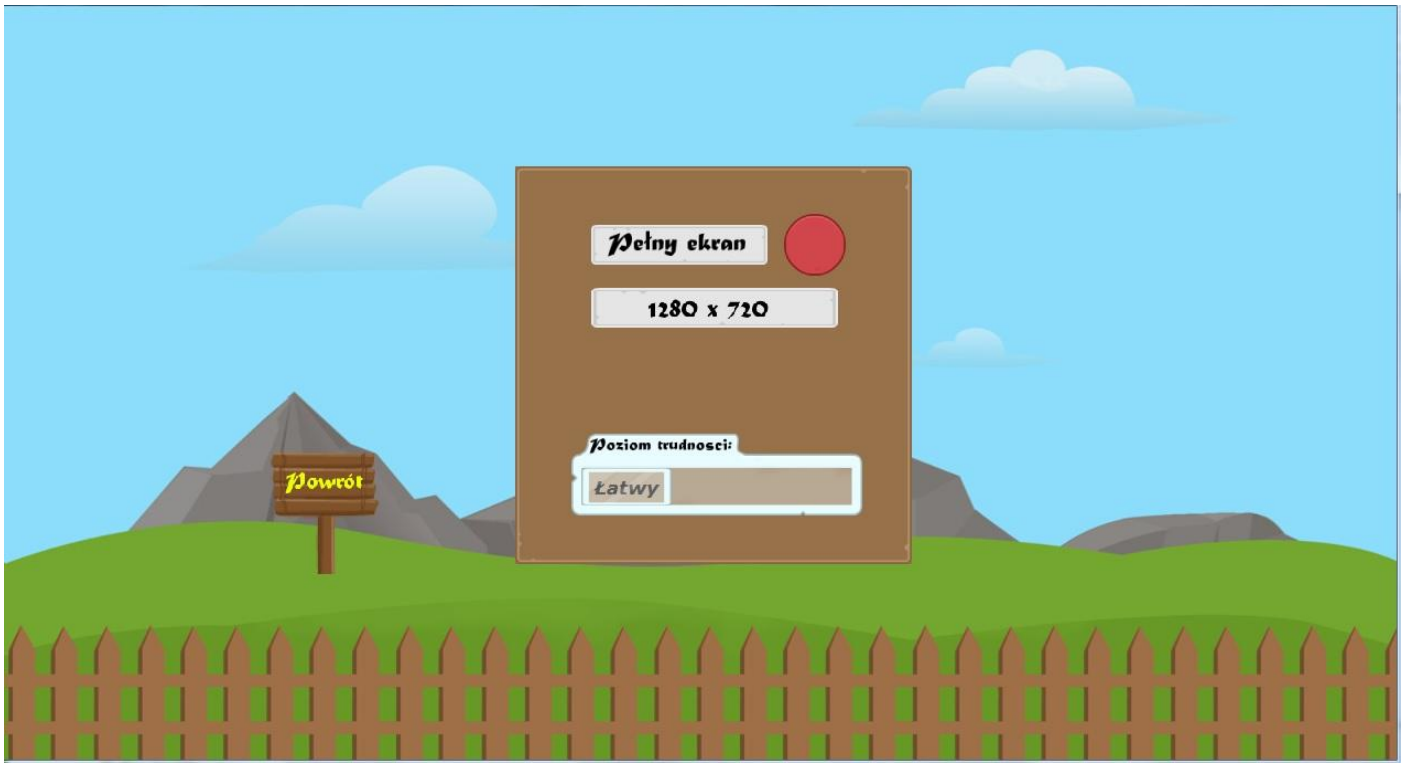
GUI zostało zaprogramowane przy użyciu pakietu Swing.

Głównym obiektem gry jest klasa Gra dziedzicząca po JFrame. Wszystkie elementy graficzne znajdują się w tym obiekcie. Klasa ta ma zdefiniowany własny wygląd kursora, którego wygląd zmienia się podczas klikania, przyciskania, czy uwalniania przycisków myszy. Używana jest do tego klasa EventKursor.

Każdy ekran gry, tj. menu (Menu), ekran rozgrywki (Granie), czy ustawień (EkranUstawien) jest klasą dziedziczącą z abstrakcyjnej klasy Ekran, która dziedziczy z kolei po klasie JPanel. Każdy z tych ekranów wyświetla się na obiekcie Gra. Wszystkie elementy GUI, takie jak przyciski, obrazki, czy labele znajdują się na tych ekranach. Ekrany te implementują interfejs ActionListener w celu nasłuchiwania kliknięć myszą.

Na ekranie Menu widać przyciski zajmujące się rozpoczęciem nowej gry (przejście do ekranu Granie), wczytaniem już istniejącego zapisu, zmianą ustawień (przejście do ekranu EkranUstawien) oraz wyjścia z gry. W przypadku próby wczytania gry, jeśli plik ze stanem gry nie istnieje, aplikacja pokazuje okno dialogowe informujące o tym, iż nie udało się wczytać gry.

Na ekranie Granie wyświetlane zostają JLabele z ilością zasobów graczy oraz ich przyrostem, zamki graczy, ich wieże oraz mury. Wyświetlone są karty w odpowiednich taliach. Pojawiają się również przyciski (obiekty klasy rozszerzającej JButton) zapisu gry, powrotu do menu, pominięcia tury i wyrzucenia kart. Po kliknięciu przycisku zapisu gry pojawia się okno dialogowe informujące o pomyślnym zapisaniu gry. W przypadku naciśnięcia przycisku powrotu do menu wyświetlane jest kolejne okno dialogowe pytające użytkownika czy na pewno chce opuścić grę i wrócić do menu.



Na ekranie EkranUstawien znajdują się przyciski odpowiadające za włączenie pełnego ekranu, zmianę rozdzielczości, poziomu trudności gry oraz powrotu do menu.