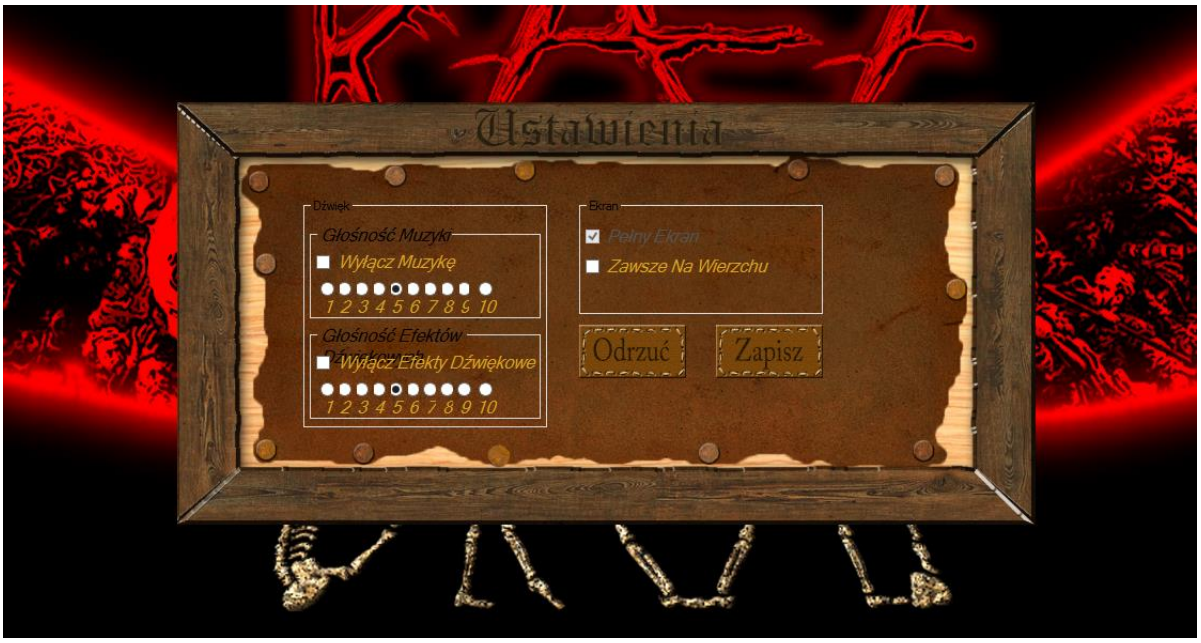
Dokumentacja projektu zespołowego
Gra RPG „Rzeź Po Grób”

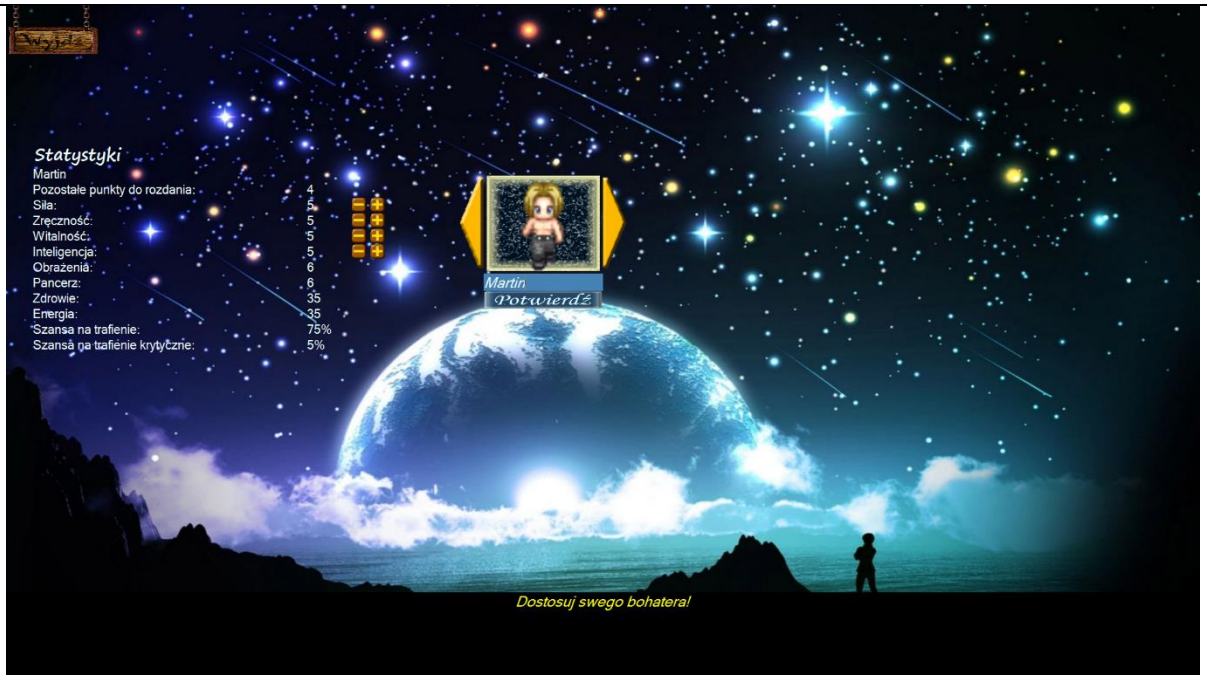
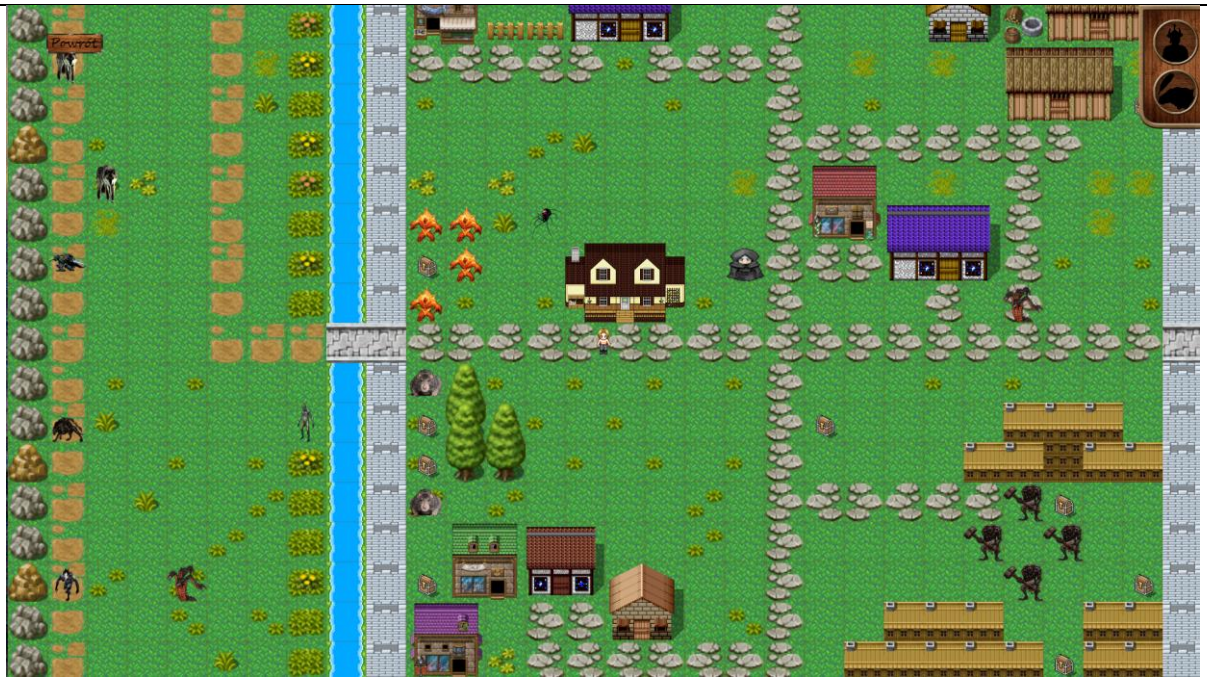
Marcin Bury
Tomasz Rosiak
Martin Pradler
Anna Snopek
Informatyka, IO, gr. I

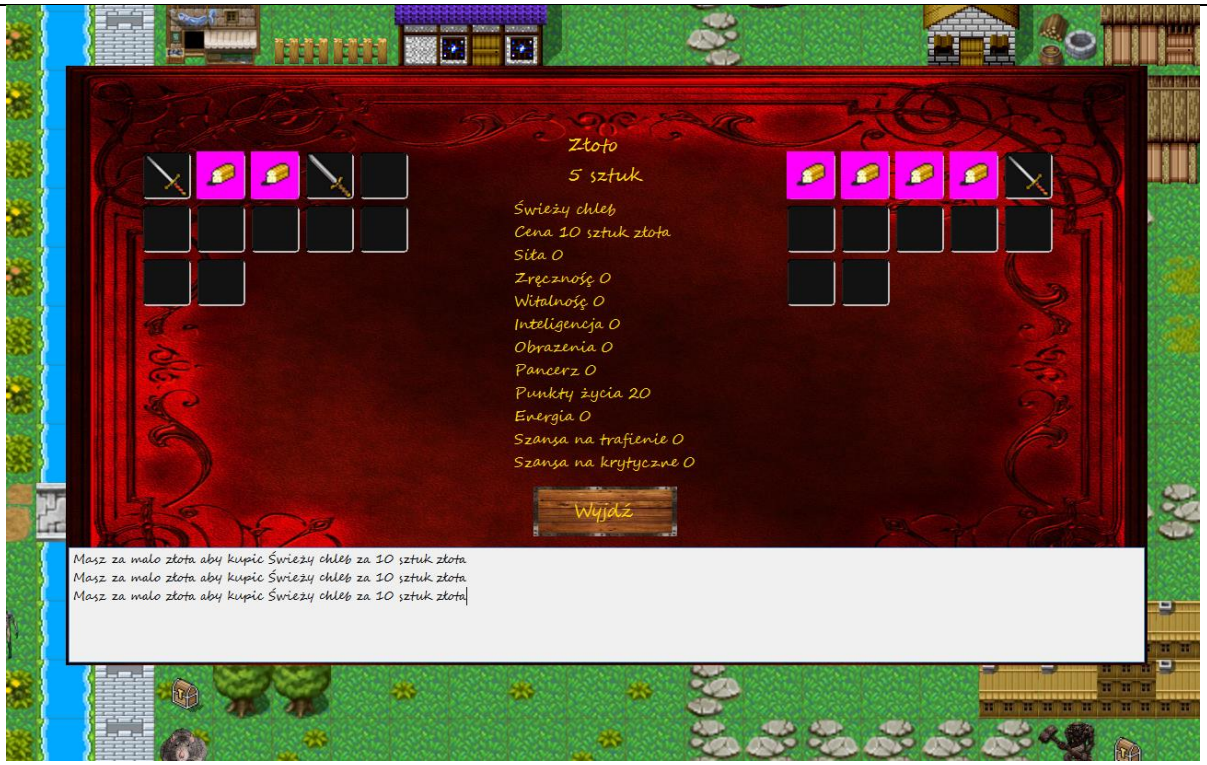
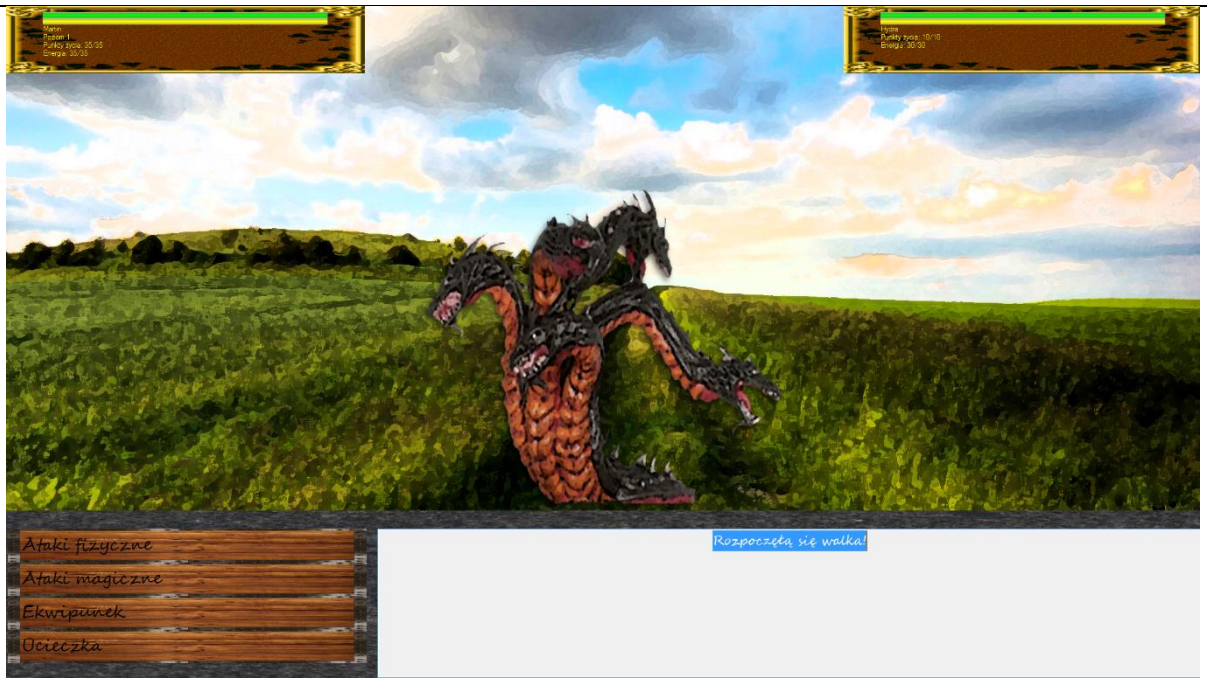
I Opis funkcjonalności zaimplementowanych w projekcie.

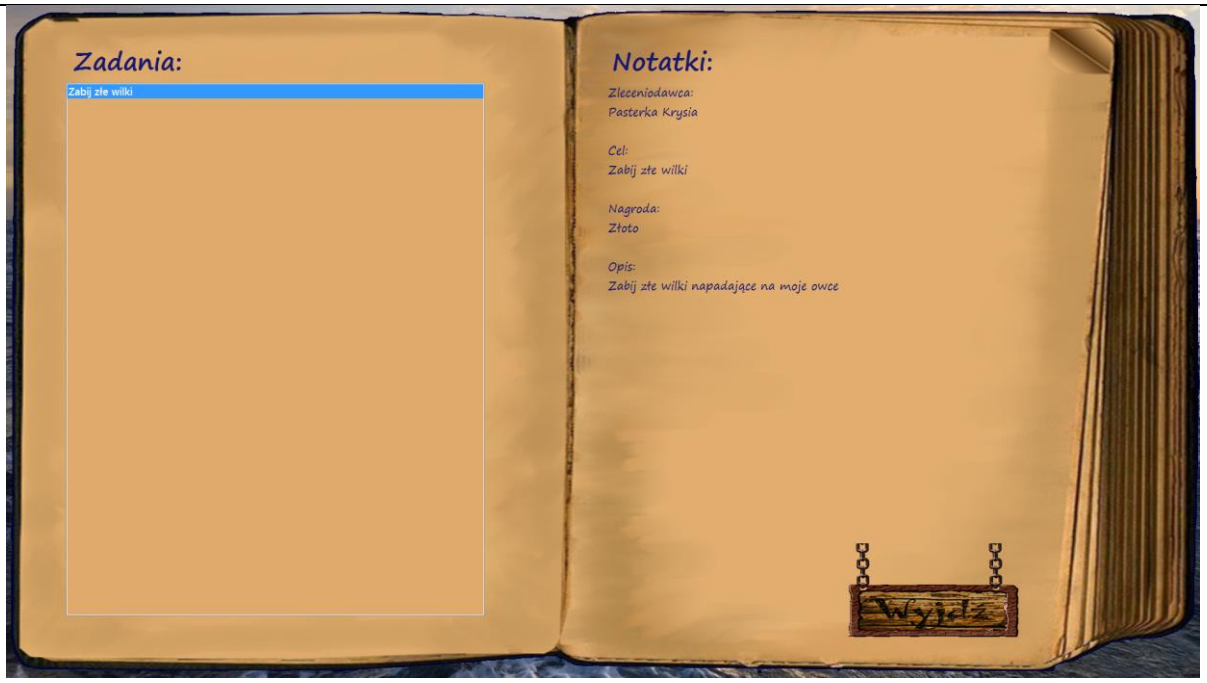
Menu główne	**Wiej** - wyjście z gry **Opcje:** Opcje związane z dźwiękiem; Opcje związane z ustawieniami ekranu; **Do boju** - rozpoczęcie rozgrywki
Ekran statystyk	Po rozpoczęciu rozgrywki pojawia się ekran, w którym znajdziemy statystyki które można zmodyfikować, rozdając wolne punkty do poszczególnych statystyk. Możliwość jest również wyboru rodzaju wyglądu naszej postaci oraz wpisać jej imię.
Ekran rozgrywki	Nasza wybrana postać pojawia się na mapie, może się poruszać za pomocą klawiszy WSAD według ogólnie przyjętych norm sterowania. Można wchodzić w interakcje z postaciami po przez wejście na nie postacią. Jeśli jest to postać niezależna, rozpocznie się dialog między postaciami lub pojawi się możliwość handlu. Natomiast jeżeli jest to przeciwnik, rozpocznie się walka. W prawym górnym rogu są możliwe do wyboru otwarcia ekwipunku gracza i dziennik zadań.
Ekran handlu	Z prawej strony znajduje się ekwipunek gracza, z lewej zaś przedmioty które możemy nabyć. Za pomocą pojedynczego kliknięcia na przedmiot wyświetlają się jego statystyki oraz cena. Metoda drag and drop możemy sprzedać przedmioty oraz kupić jeżeli posiadamy odpowiednią ilość.
Ekran walki	Ekran walki przedstawia pojedynek gracza z napotkanym przeciwnikiem. W lewym górnym roku znajdują się obecne statystyki dotyczące gracza, aktualny poziom postaci, punkty życia oraz energia. W lewym górnym znajdują się statystyki przeciwnika, jego nazwa, punkty życia oraz energia. Upływ ich jest reprezentowany za pomocą zmiany paska obok statystyki na czerwony oraz zmniejszenie ich wartości. Po każdej walce statystyki protagonisty są zapisywane. W lewym dolnym rogu znajduje się menu akcji w którym do wyboru mamy, ataki fizyczne, ataki magiczne, ekwipunek. Dwa pierwsze wymienione przeze mnie są wykorzystywane do zadawania obrażeń przeciwnikowi, w taki sposób by jego punkty zdrowia spadły poniżej zera. W ekwipunku znajdują się przedmioty które możemy użyć kosztem tury, są to rzeczy odnawiające życie oraz energię. Po prawej stronie u dołu jest ekran przebiegu walki w wersji tekstowej.

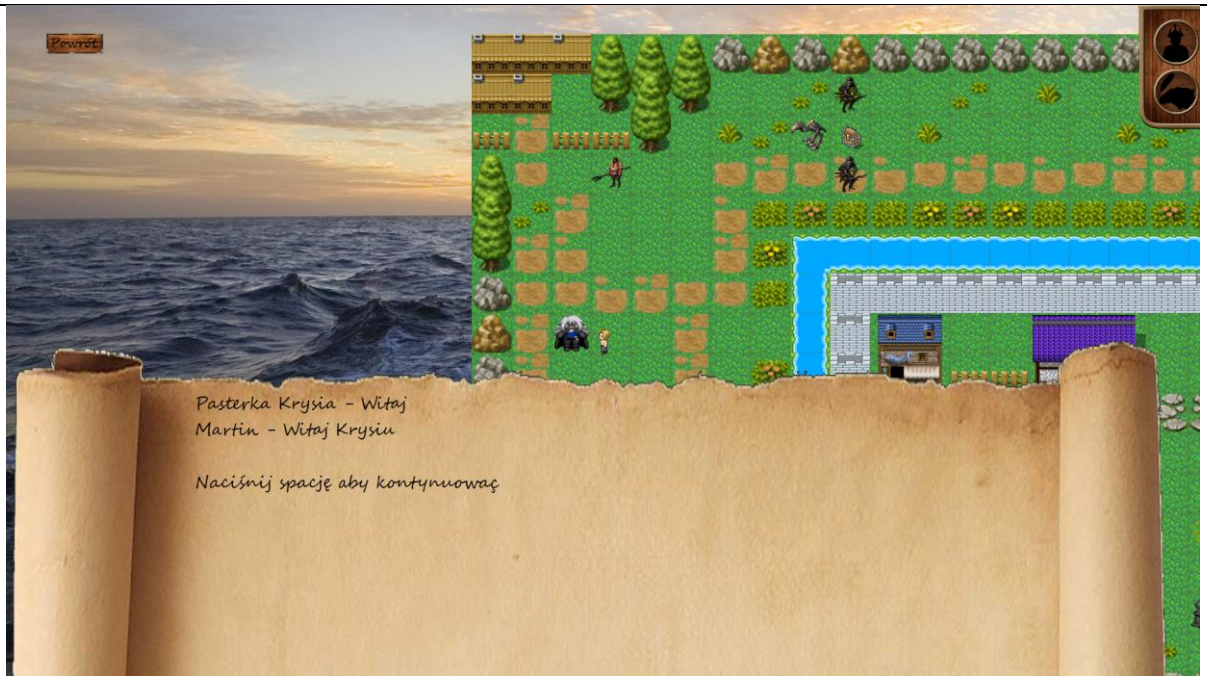
	Pośrodku ekranu znajduje obraz przeciwnika który go reprezentuje, a w tle znajdują się losowo generowane grafiki dodające klimatu polu walki.
Ekran ekwipunku	Ekran ten służy do przeglądu posiadanych przedmiotów, które znajdują się po prawej stronie w odpowiednim polu. Zmienić ekwipunek możemy za pomocą przeciągania przedmiotu na odpowiednie pole metodą drag and drop. Z lewej strony przedstawione są nasze wszystkie aktualne statystyki, które po zdobyciu poziomu możemy rozwinąć poprzez manipulację przy guzikach dodawania punktów otrzymanych za nowy poziom.
Ekran dziennika zadań	Znajdują się w nim aktualne zadania zlecone przez postacie. Z lewej strony znajduje się ich spis, po jego kliknięciu z prawej strony wyświetlane są szczegółowe informacje.
Ekran dialogów	Reprezentuje rozmowę między graczem a postaciami napotkanymi na mapie

Menu główne	 

Ekran statystyk	
Ekran rozgrywki	

Ekran handlu	
Ekran walki	

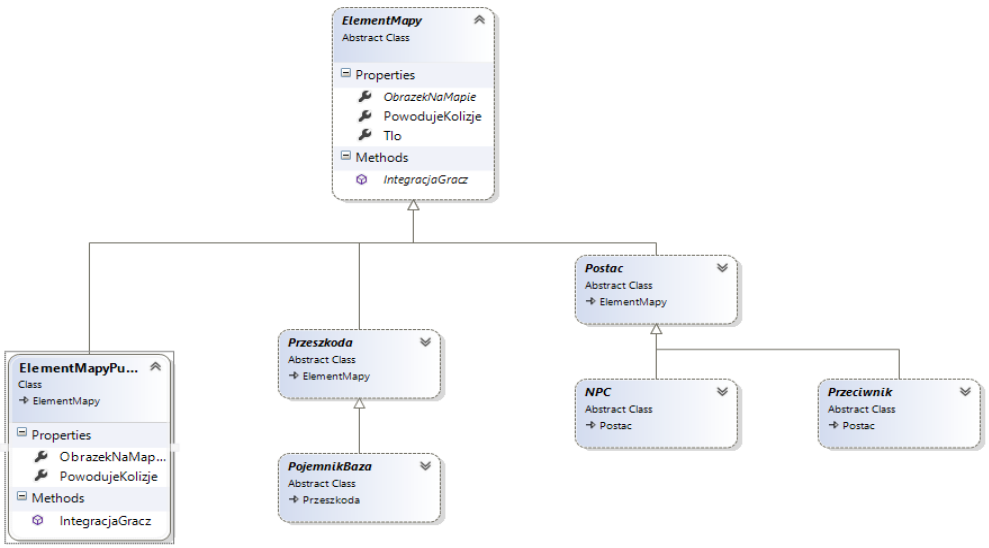
Ekran ekwipunku	
Ekran dziennika zadań	

Ekran dialogów	

II Analiza kluczowych elementów projektu oraz sposobu ich implementacji

Format mapy i mechanizm jej wczytywania

Mapa reprezentowana jest przez klasę Obszar, zawiera ona dwuwymiarową tablicę typu ElementMapy. Każdy element tej tablicy reprezentuje jedną komórkę mapy. Klasa ElementMapy jest klasą bazową dla wszystkich obiektów, które mogą znaleźć się na mapie.

Hierarchia klas dziedziczących po elemencie mapy	
Obszar mapy zapisany jest w pliku txt, sformatowanym w	• w pierwszej linii rozmiar mapy (ilość wierszy, kolumn) oraz współrzędne startowe gracza (numer wiersza, numer kolumny) • Poniżej znajduje się macierz elementów w rozmiarze zdefiniowanym w pierwszej linii pliku

następujący sposób:	• Każdy element macierzy ma format: • (obiekt:rodzajobiektu;tlo/adresGrafikiBędącejTłemDanegoElementu/nazwa.png) wybrana grafika plus tło • (obiekt:rodzajobiektu) pojawia się grafika obiektu bez tła • (tlo/adresGrafikiBędącejTłemDanegoElementu/nazwa.png) grafika tła bez obiektu Każdy element niezdefiniowany jako obiekt, nie powoduje kolizji z graczem.

Fragment pliku txt z mapą gry

zywiolakZiemi.cs zywiolakOgnia.cs zywiolakWody.cs mapa.txt

```
40 40 3 3
(obiekt:domgora                                                    )    (obiekt:domgora
(obiekt:domgora                                                    )    (obiekt:domgora
(obiekt:obejscie1                                                  )    (tlo:Resources\Mapy\sciezka.png
(obiekt:Tree1_1;tlo:Resources\Mapy\grass.png                       )    (tlo:Resources\Mapy\sciezka3.png
(obiekt:Tree1_1_2;tlo:Resources\Mapy\grass.png                     )    (tlo:Resources\Mapy\roslina1_4.png
(obiekt:Tree1_2;tlo:Resources\Mapy\grass.png                       )    (tlo:Resources\Mapy\sciezka.png
(obiekt:skala1                                                     )    (tlo:Resources\Mapy\sciezka3.png
(obiekt:skala2                                                     )    (tlo:Resources\Mapy\sciezka.png
(obiekt:skala1                                                     )    (tlo:Resources\Mapy\sciezka.png
(obiekt:skala1                                                     )    (tlo:Resources\Mapy\sciezka3.png;przeciwnik:WilkKrysia
(obiekt:skala1                                                     )    (tlo:Resources\Mapy\sciezka.png
(obiekt:skala2                                                     )    (tlo:Resources\Mapy\sciezka2.png
(obiekt:skala1                                                     )    (tlo:Resources\Mapy\sciezka.png
(obiekt:skala1                                                     )    (tlo:Resources\Mapy\sciezka3.png
(obiekt:skala1                                                     )    (przeciwnik:goblinzabojca;tlo:Resources\Mapy\sciezka.png
(obiekt:skala1                                                     )    (tlo:Resources\Mapy\sciezka3.png
(obiekt:skala1                                                     )    (tlo:Resources\Mapy\sciezka.png
(obiekt:skala1                                                     )    (tlo:Resources\Mapy\sciezka2.png
(obiekt:skala1                                                     )    (przeciwnik:gzwiadowca;tlo:Resources\Mapy\sciezka.png
(obiekt:skala2                                                     )    (tlo:Resources\Mapy\sciezka3.png
(obiekt:skala1                                                     )    (tlo:Resources\Mapy\sciezka.png
(obiekt:skala1                                                     )    (tlo:Resources\Mapy\sciezka2.png
(obiekt:skala2                                                     )    (przeciwnik:argul;tlo:Resources\Mapy\sciezka.png
(obiekt:skala1                                                     )    (tlo:Resources\Mapy\sciezka3.png
(obiekt:skala1                                                     )    (tlo:Resources\Mapy\sciezka.png
(obiekt:skala1                                                     )    (tlo:Resources\Mapy\sciezka3.png
(obiekt:skala1                                                     )    (przeciwnik:gwojownik;tlo:Resources\Mapy\sciezka.png
(obiekt:skala1                                                     )    (tlo:Resources\Mapy\sciezka.png
(obiekt:skala2                                                     )    (tlo:Resources\Mapy\sciezka2.png
(obiekt:skala1                                                     )    (tlo:Resources\Mapy\sciezka.png
(obiekt:skala1                                                     )    (tlo:Resources\Mapy\sciezka3.png
(obiekt:skala1                                                     )    (tlo:Resources\Mapy\sciezka2.png
(obiekt:skala1                                                     )    (tlo:Resources\Mapy\grass.png
(obiekt:skala1                                                     )    (tlo:Resources\Mapy\roslina3.png
(obiekt:Tree1_1;tlo:Resources\Mapy\grass.png                       )    (przeciwnik:mis;tlo:Resources\Mapy\grass.png
(obiekt:Tree1_2;tlo:Resources\Mapy\grass.png                       )    (tlo:Resources\Mapy\grass.png
(obiekt:Tree1_1;tlo:Resources\Mapy\grass.png                       )    (tlo:Resources\Mapy\roslina3.png ;obiekt:skrzynia1
(obiekt:Tree1_2;tlo:Resources\Mapy\grass.png                       )    (obiekt:Tree1_1;tlo:Resources\Mapy\grass.png
(obiekt:Tree1_1;tlo:Resources\Mapy\grass.png                       )    (obiekt:Tree1_1_2;tlo:Resources\Mapy\grass.png
(obiekt:Tree1_2;tlo:Resources\Mapy\grass.png                       )    (obiekt:Tree1_2;tlo:Resources\Mapy\roslina3.png
```

Fragment kodu odpowiedzialny za wczytywanie mapy:
Wczytywanie mapy bazuje na mechanizmie refleksji. Odczytujemy jakie typy dziedziczą po klasie element bazowy i na podstawie wybranego elementu macierzy tworzymy instancję odpowiedniej klasy

```
public static Obszar WczytajObszar(string nazwaObszaru)
{
    if (!obszary.ContainsKey(nazwaObszaru))
    {
        obszary[nazwaObszaru] = Wczytaj(nazwaObszaru);
    }
    return obszary[nazwaObszaru];
}
private static Dictionary<string, Type> przeszkody = new Dictionary<string, Type>();
private static Dictionary<string, Type> postacie = new Dictionary<string, Type>();
private static Dictionary<string, Type> przeciwnicy = new Dictionary<string, Type>();
static ManagerObszarow()
{
    przeszkody = Assembly.GetExecutingAssembly().GetTypes().Where(x => !x.IsAbstract && x.IsSubclassOf(typeof(Przeszkoda))).ToDictionary(x => x.Name.ToLower(), x => x);//Pobieramy wszystkie klasy dziedziczące po klasie Przeszkoda
    postacie = Assembly.GetExecutingAssembly().GetTypes().Where(x => !x.IsAbstract && x.IsSubclassOf(typeof(NPC))).ToDictionary(x => x.Name.ToLower(), x => x);//Pobieramy wszystkie klasy dziedziczące po klasie NPC
    przeciwnicy = Assembly.GetExecutingAssembly().GetTypes().Where(x => !x.IsAbstract && x.IsSubclassOf(typeof(Przeciwnik))).ToDictionary(x => x.Name.ToLower(), x => x);//Pobieramy wszystkie klasy dziedziczące po klasie Przeciwnik
}
private static ElementMapy StworzObiekt(string p)
{
    return (ElementMapy)Activator.CreateInstance(przeszkody[p]);
}

private static ElementMapy StworzPrzeciwnika(string p)
{
    return (ElementMapy)Activator.CreateInstance(przeciwnicy[p]);
}
private static ElementMapy StworzNpc(string p)
{
    return (ElementMapy)Activator.CreateInstance(postacie[p]);
}
```

Renderowanie i poruszanie się postaci na mapie

Klasa EkranGry zawiera wszystkie mechanizmy odpowiedzialne za renderowanie i poruszanie się postaci na mapie. Ponieważ aplikacja oparta jest na Windows Forms, klasa EkranGry dziedziczy po klasie Form (klasa bazowa dla wszystkich okien w systemie windows). W klasie EkranGry zaimplementowaliśmy timer, który co określony czas wywołuje metodę odświeżającą aktualny stan gry. Metoda ta uruchamiana jest co ok. 30 ms i na podstawie wciśniętych przez gracza przycisków, aktualizuje stan w jakim znajduje się gra i decyduje czy należy przerysować wyświetlany ekran.

```
private void timerPrzeplywCzasu_Tick(object sender, EventArgs e)
{
    bool odswierzamy=false;
    if (!string.IsNullOrEmpty(Komunikat))
    {
        if (CzasWyswietlanieKomunikatu > 0)
        {
            CzasWyswietlanieKomunikatu -= timerPrzeplywCzasu.Interval;
        }
        if (CzasWyswietlanieKomunikatu <= 0)
        {
            odswierzamy = true;
            Komunikat = null;
            CzasWyswietlanieKomunikatu = 0;
        }
    }
    bool bylaKolizja = false;
    Ruch? kierunekRuchuGracz=null;

    int px=0;
    int py=0;
    if(CzyWczysnieto(Keys.Up))
    {
        kierunekRuchuGracz = Ruch.Gora;
        py-=SzybkoscRuchow;
        Gra.gracz.AktualnyObrazek = "góra.gif";
        idxklatki++;
    }
    else if(CzyWczysnieto(Keys.Down))
    {
        kierunekRuchuGracz = Ruch.Dol;
          py+=SzybkoscRuchow;
          Gra.gracz.AktualnyObrazek = "dół.gif";
          idxklatki++;
    }
    else if(CzyWczysnieto(Keys.Left))
    {
        kierunekRuchuGracz = Ruch.Lewo;
          px-=SzybkoscRuchow;
          Gra.gracz.AktualnyObrazek = "lewo.gif";
          idxklatki++;
    }
    else if (CzyWczysnieto(Keys.Right))
    {
        kierunekRuchuGracz = Ruch.Prawo;
          px+=SzybkoscRuchow;
          Gra.gracz.AktualnyObrazek = "prawo.gif";
          idxklatki++;
    }
    if (!kierunekRuchuGracz.HasValue)
```

```
if (!kierunekRuchuGracz.HasValue)
{
    Gra.gracz.AktualnyObrazek = null;
}
else
{
    Rectangle graczmapa = new Rectangle(pozycjaGracza[0] + px, pozycjaGracza[1] + py, Gra.gracz.Szerokosc, Gra.gracz.Wysokosc);//gdzie były gracz gdyby się przesunął
    for (int i = 0; i < obszarGry.Mapa.GetLength(0); i++)
    {
        for (int j = 0; j < obszarGry.Mapa.GetLength(1); j++)
        {
            ElementMapy element = obszarGry.Mapa[i, j];
            if (element == null || !element.PowodujeKolizje)
            {
                continue;
            }
            Rectangle pozycjaelemnt = new Rectangle(j * obszarGry.Rozmiar, i * obszarGry.Rozmiar, obszarGry.Rozmiar, obszarGry.Rozmiar);
            if (graczmapa.IntersectsWith(pozycjaelemnt))
            {
                if (kierunekRuchuGracz == Ruch.Lewo && pozycjaelemnt.Right >= graczmapa.Left)
                {
                    bylaKolizja = true;
                }
                else if (kierunekRuchuGracz == Ruch.Prawo && pozycjaelemnt.Left <= graczmapa.Right)
                {
                    bylaKolizja = true;
                }
                else if (kierunekRuchuGracz == Ruch.Gora && pozycjaelemnt.Bottom >= graczmapa.Top)
                {
                    bylaKolizja = true;
                }
                else if (kierunekRuchuGracz == Ruch.Dol && pozycjaelemnt.Top <= graczmapa.Bottom)
                {
                    bylaKolizja = true;
                }
                if (bylaKolizja)
                {
                    timerPrzeplywCzasu.Stop();
                    element.IntegracjaGracz(Gra.gracz, i, j, this);
                    timerPrzeplywCzasu.Start();
                    break;
                }
            }
        }
        if (bylaKolizja)
        {
            break;
        }
    }
            if (bylaKolizja)
            {
                break;
            }
        }
        if (!bylaKolizja)
        {
            pozycjaGracza[0] += px;
            pozycjaGracza[1] += py;
        }
        odswierzamy = true;
    }
    if (odswierzamy)
    {
        this.Invalidate();
    }
}
```

Renderowanie ekranu gry oparte jest na zdarzeniu Paint klasy Form. W celu poprawy szybkości rysowania metoda ta wylicza które elementy mają być narysowane i pomija te, które znajdą się poza ekranem. Metoda przyjmuje założenie że postać gracza renderowana jest zawsze na środku ekranu, a tło jest odpowiednio przesuwane na podstawie wciśniętych przez gracza klawiszy.

```
private void Form1_Paint(object sender, PaintEventArgs e)
{
    int xgracz = Width / 2 - Gra.gracz.Szerokosc / 2;
    int ygracz = Height / 2 - Gra.gracz.Wysokosc / 2;
    Graphics g = e.Graphics;
    for (int i = 0; i < obszarGry.Mapa.GetLength(0); i++)
    {
        for (int j = 0; j < obszarGry.Mapa.GetLength(1); j++)
        {
            var x = j * obszarGry.Rozmiar +xgracz -pozycjaGracza[0];
            int y = i * obszarGry.Rozmiar + ygracz - pozycjaGracza[1];

            Rectangle r = new Rectangle(x,y, obszarGry.Rozmiar, obszarGry.Rozmiar);
            if(!r.IntersectsWith(e.ClipRectangle))
            {
                continue;// nie rysujemy elelementów które nie są widoczne
            }
            if (obszarGry.Mapa[i, j]==null)
            {
                continue;
            }
            if (obszarGry.Mapa[i, j].Tlo != null)
            {
                using (var ia = new ImageAttributes())
                {
                    ia.SetWrapMode(WrapMode.TileFlipXY);
                    g.DrawImage(MenagerZasobow.PobierzBitmape(obszarGry.Mapa[i, j].Tlo), r, 0, 0, obszarGry.Rozmiar, obszarGry.Rozmiar, GraphicsUnit.Pixel, ia);
                }
            }
            if (obszarGry.Mapa[i, j].ObrazekNaMapie != null)
            {
                g.DrawImage(MenagerZasobow.PobierzBitmape(obszarGry.Mapa[i, j].ObrazekNaMapie), r);
            }
        }
    }
    Image graczimg = MenagerZasobow.PobierzBitmape(Gra.gracz.ObrazekNaMapie + (Gra.gracz.AktualnyObrazek ?? "dół.png"));
    FrameDimension dimension = new FrameDimension(graczimg.FrameDimensionsList[0]);
    int frameCount = graczimg.GetFrameCount(dimension);
    if(idxklatki>=frameCount)
    {
        idxklatki = 0;
    }

    graczimg.SelectActiveFrame(dimension, idxklatki);

    Rectangle pozycjagracz = new Rectangle(xgracz, ygracz, Gra.gracz.Szerokosc, Gra.gracz.Wysokosc);
    g.DrawImage(graczimg, pozycjagracz);
    if(!string.IsNullOrEmpty(Komunikat))
    {
        Rectangle pozycjatekstu=new Rectangle(0,(int)(Height*0.9),Width,(int)(Height*0.1f));
        g.DrawString(Komunikat, new Font("Segoe Script", 20), new SolidBrush(System.Drawing.Color.Gold), pozycjatekstu);
    }
}
```

III Umiejętności postaci i mechanizm ich wczytywania

Umiejętności to specjalne klasy definiujące możliwe ataki na przeciwnika, np. kula ognia, pchnięcie itp. Każda umiejętność reprezentowana jest przez klasę dziedziczącą po klasie Umiejetnosc.

Klasa Umiejetnosc

1. Właściwości:

Nazwa - nazwa umiejętności, musi być implementowana w każdej z klas potomnych

Magiczna - czy daną umiejętność traktujemy jako magiczną, musi być implementowana w każdej z klas potomnych

KosztEnergii - koszt użycia danej umiejętności, domyślnie 0, klasy potomne mogą ją nadpisać

2. Metody:

Wykonaj - metoda, w której implementujemy działanie danej umiejętności, każda klasa potomna musi ją implementować

ZaplaćZaUzycie - metoda implementująca koszt użycia umiejętności

Atak - Metoda przeprowadzająca atak, decyduje czy umiejętność zadziałała, modyfikuje atakującego i cel ataku

CzyTrafiono - metoda decydująca czy atak się powiódł

JestDostepna - zwraca informację czy gracz może użyć danej umiejętności, musi być implementowana przez każdą klasę potomną

3. Opis działania
Każdy z przeciwników gracza ma z góry zdefiniowane jakie ma umiejętności, natomiast gracz podczas każdej tury walki ma wyliczane jakie umiejętności może użyć w danym momencie.
Wczytywanie umiejętności jest dynamiczne, bazuje na mechanizmie refleksji. Z aktualnego pakietu odczytujemy jakie klasy są w nim zdefiniowane, pobieramy wszystkie typy dziedziczące po klasie Umiejętnosc. Tworzymy na podstawie tych typów instancję tych klas, wywołujemy metodę JestDostępna dla gracza, jeżeli zwróci true, to gracz może użyć tej umiejętności.

```
public override List<Umiejetnosc> Umiejetnosci()
{
    List<Umiejetnosc> wynik = new List<Umiejetnosc>();
    List<Type> typy = Assembly.GetExecutingAssembly().GetTypes().Where(x => x.IsSubclassOf(typeof(Umiejetnosc))).ToList();//Pobieramy wszystkie klasy dziedziczące po klasie Umiejetnosc
    foreach(Type t in typy)
    {
      Umiejetnosc testowana=(Umiejetnosc)  Activator.CreateInstance(t);//Tworzymy instancje klasy na podstawie typu - używamy konstruktora domyślnego tej klasy
        if(testowana.JestDostepna(this))
        {
            wynik.Add(testowana);
        }
    }
    return wynik;

}
```