Gymnázium a Střední průmyslová škola elektrotechniky a informatiky, Frenštát pod Radhoštěm, příspěvková organizace

Praktická zkouška z Programování

Console Dungeon

Jméno: Jan Žňava
Třída: T4A
Rok: 2024/2025
Téma: Objektově orientované programování

Obsah

Zadání: 2
Úvod 3
Rozdělení tříd 3
Třída character 4
Třída hero 6
Třída warrior 9
Příklady dalších hero tříd 11
Třída enemy 12
Třída rat 13
Příklady dalších enemy tříd 14
Hlavní program 16
Závěr 22

Zadání:

Vytvoř aplikaci v jazyce C#, která bude obsahovat:

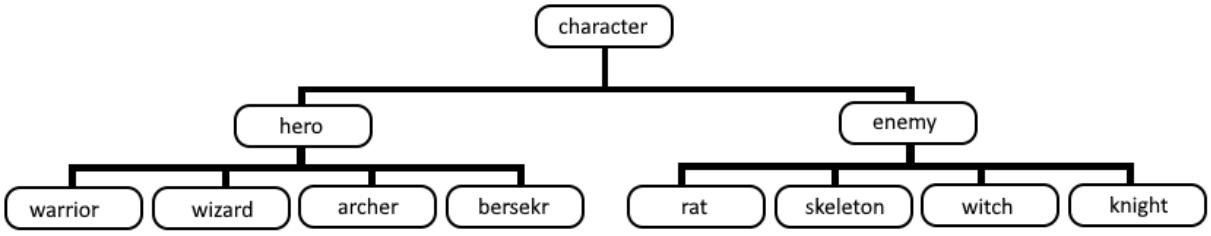
- Dědičnost o minimálně třech stupních
- Minimálně 5 objektových tříd
- Abstraktní třídu
- Virtuální metody
- Vlastnosti
- Aplikaci, která využije všechny (mimo abstraktní) objektové třídy

Úvod

Jedná se o bojovou hru v konzoli, kde jako hrdina musíte projít několik místností a porazit několik nepřátel. OOP jsem použil právě na rozdělení postav a na naprogramování jejich schopností a vlastností.

Rozdělení tříd

Hlavní abstraktní třída je character, vytváří se zní třídy hero a enemy. Třída hero slouží pro postavy hráče, obsahuje navíc jeho možné akce, zatímco třída enemy slouží hlavně pro naprogramování chování daného nepřítele. Z třídy hero se následně vytváří třídy daných hrdinů a ze třídy enemy se vytváří třídy daných nepřátel.



Třída character

Abstraktní třída obsahuje základní proměnné, které využívají všechny dědičné třídy:

- name – název postavy
- texture – je to string array, která obsahuje text představující grafiku postavy
- health – životy postavy
- max_health – maximální životy
- armor – brnění postavy
- max_armor – maximální brnění
- damage – poškození které postava udělí nepříteli
- alive – určuje, jestli postava žije
- position – pozice na x souřadnice, určuje, kde se postava nachází a kde se má vykreslit
- stamina – určuje kolik akcí může postava udělat

Konstruktor pro vytvoření a druhý pro zkopírování dat z již vytvořené stejné třídy, toto bude později více vysvětleno. Dále virtuální funkci TakeDamage která se vyvolává od jiného bojovníka, který uvede poškození, které si má daná postava ubrat. Nejčastější je použití takové, že chceme, aby se nejdřív poškození ubralo od brnění (ubírá se poloviční poškození) a následně až životy a taky se případně aktualizuje proměnná alive, která značí, jestli je postava živá.

```
Počet odkazů: 9
public abstract class character
{
    public string name;
    public string[] texture; //17 height x 6 width [char]
    public int health;
    public int max_health;
    public int armor;
    public int max_armor;
    public int damage;
    public bool alive = true;
    public int position;
    public int stamina = 20;

    Počet odkazů: 2
    public character(string name, string[] texture, int health, int armor, int damage)
    {
        this.name = name;
        this.texture = texture;
        this.health = health;
        this.max_health = health;
        this.armor = armor;
        this.max_armor = armor;
        this.damage = damage;
    }

    Počet odkazů: 2
    public character(character ch)
    {
        this.name = ch.name;
        this.texture = ch.texture;
        this.health = ch.health;
        this.max_health = ch.health;
        this.armor = ch.armor;
        this.max_armor = ch.armor;
        this.damage = ch.damage;
    }

    Počet odkazů: 12
    public virtual void TakeDamage(int damage)
    {
        if (armor > 0)
        {
            armor -= damage / 2;
            if (armor <= 0)
                armor = 0;
        }
        else
        {
            health -= damage;
            if (health <= 0)
                alive = false;
        }
    }

}
```

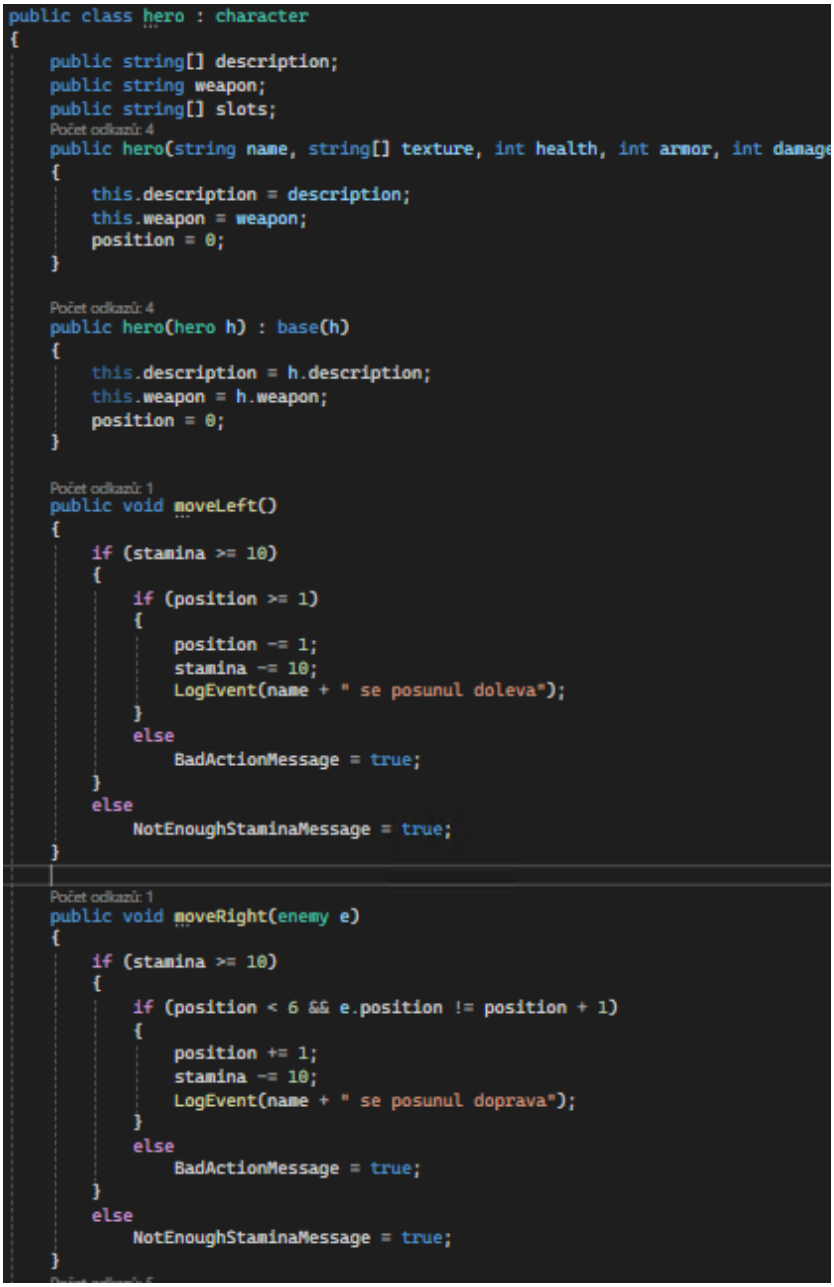
Třída hero

Třída vytvořena ze třídy character, obsahují navíc proměnné:

- description – string array obsahují stručný popis postavy
- weapon – název zbraně, kterou postava využívá
- slots – jaké „ikonky“ se mají potom v inventáři zobrazit jako možné akce

Má vytvořené funkce:

- moveLeft – slouží pro posun postavy doleva na bojišti
- moveRight – slouží pro posun postavy doprava na bojišti, potřebuje vědět jestli postavě nepřekáží nepřítel, proto potřebuje vstup enemy
- regenerateStamina – slouží pro obnovení staminy

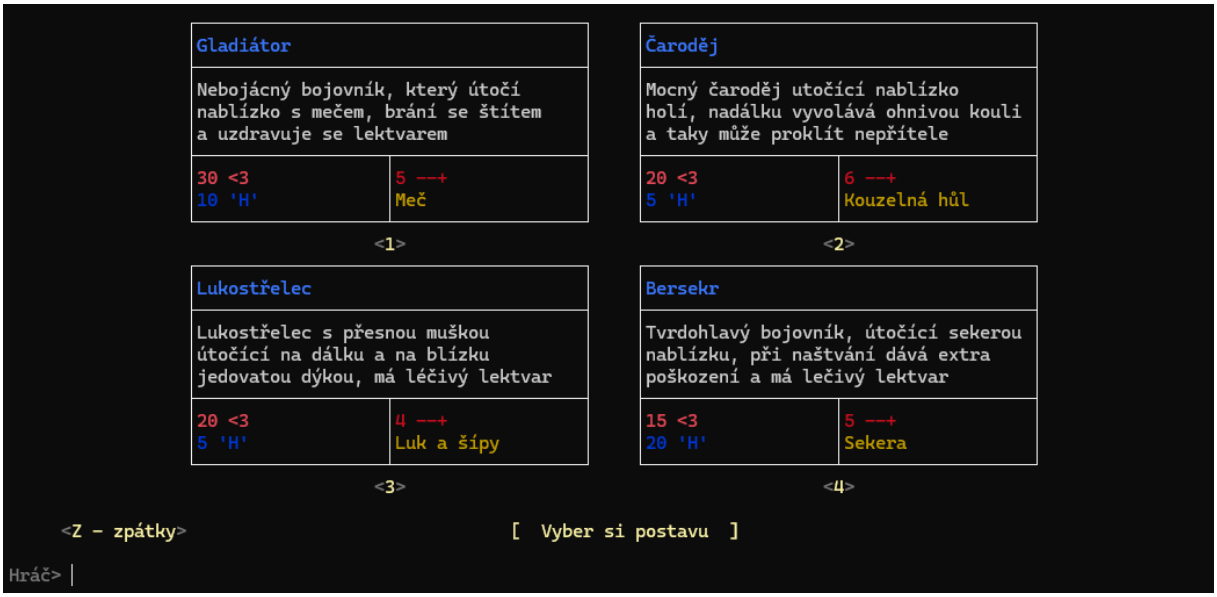


```
public class hero : character
{
    public string[] description;
    public string weapon;
    public string[] slots;
    Počet odkazů: 4
    public hero(string name, string[] texture, int health, int armor, int damage
    {
        this.description = description;
        this.weapon = weapon;
        position = 0;
    }

    Počet odkazů: 4
    public hero(hero h) : base(h)
    {
        this.description = h.description;
        this.weapon = h.weapon;
        position = 0;
    }

    Počet odkazů: 1
    public void moveLeft()
    {
        if (stamina >= 10)
        {
            if (position >= 1)
            {
                position -= 1;
                stamina -= 10;
                LogEvent(name + " se posunul doleva");
            }
            else
                BadActionMessage = true;
        }
        else
            NotEnoughStaminaMessage = true;
    }

    Počet odkazů: 1
    public void moveRight(enemy e)
    {
        if (stamina >= 10)
        {
            if (position < 6 && e.position != position + 1)
            {
                position += 1;
                stamina -= 10;
                LogEvent(name + " se posunul doprava");
            }
            else
                BadActionMessage = true;
        }
        else
            NotEnoughStaminaMessage = true;
    }
```

Dále viruální fukce, které jsou u každé postavy jiné, proto nejsou definované. Názvy attack, defend, special by se daly správně přirovnat pouze u první dědičné třídy warrior, u dalších toto schéma neplatí (např. defend může být použito i pro útok), ale přišlo mi to lepší pojmenování než jako třeba action1, action2, etc....

```
Počet odkazů: 5
public virtual void attack(enemy e)
{

}
Počet odkazů: 5
public virtual void defend(enemy e)
{

}
Počet odkazů: 5
public virtual void special(enemy e)
{

}

Počet odkazů: 2
public void regenerateStamina()
{
    stamina = 20;
}
```

Taky funkce card, pro výpis jednotlivých řádků karet s vlastnostmi bojovníka, kde její vstup je řádek, který se má vypsat, využívá se v menu pro vybrání postavy.



```
public virtual void card(int row)
{
    if (row == 0)
    {
        Console.ForegroundColor = ConsoleColor.White;
        Console.Write("  ┌──────────────────────────────┐  ");
    }

    if (row >= 1 && row != 2 && row != 6 && row < 9)
    {
        Console.ForegroundColor = ConsoleColor.White;
        Console.Write("  │");
    }

    if (row == 1)
    {
        Console.ForegroundColor = ConsoleColor.Blue;
        Console.Write(TextPad(name, 37));
    }

    if (row == 2)
    {
        Console.ForegroundColor = ConsoleColor.White;
        Console.Write("  ├──────────────────────────────┤  ");
    }

    if (row == 3)
    {
        Console.ForegroundColor = ConsoleColor.Gray;
        Console.Write(TextPad(description[0], 37));
    }
    if (row == 4)
    {
        Console.ForegroundColor = ConsoleColor.Gray;
        Console.Write(TextPad(description[1], 37));
    }
    if (row == 5)
    {
        Console.ForegroundColor = ConsoleColor.Gray;
        Console.Write(TextPad(description[2], 37));
    }

    if (row == 6)
    {
        Console.ForegroundColor = ConsoleColor.White;
        Console.Write("  ├──────────┬──────────────────┤  ");
    }
    if (row == 7)
    {
        Console.ForegroundColor = ConsoleColor.Red;
        Console.Write(TextPad(max_health + " <3", 18));

        Console.ForegroundColor = ConsoleColor.White;
        Console.Write("│");

        Console.ForegroundColor = ConsoleColor.DarkRed;
        Console.Write(TextPad(damage + " --+", 18));
    }

    if (row == 8)
    {
        Console.ForegroundColor = ConsoleColor.DarkBlue;
        Console.Write(TextPad(armor + " 'H'", 18));

        Console.ForegroundColor = ConsoleColor.White;
        Console.Write("│");

        Console.ForegroundColor = ConsoleColor.DarkYellow;
        Console.Write(TextPad(weapon, 18));
    }

    if (row == 9)
    {
        Console.ForegroundColor = ConsoleColor.White;
        Console.Write("  └──────────┴──────────────────┘  ");
    }

    if (row >= 1 && row != 2 && row != 6 && row < 9)
    {
        Console.ForegroundColor = ConsoleColor.White;
        Console.Write("  │  ");
    }

    Console.ForegroundColor = ConsoleColor.White;
}
```

Třída warrior

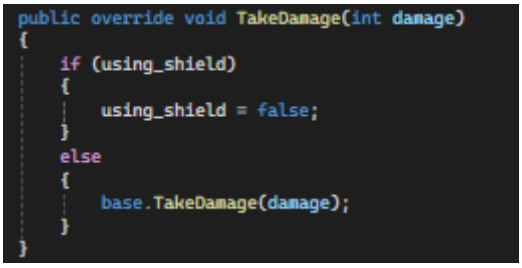
Je dědičná třída od hero a má vlastní proměnnou using_shield, který se používá ve změněné funkci TakeDamage, pro použití štítu, také v konstruktorech nastavujeme slots proměnnou, aby se zobrazovaly správné akce, které může postava warrior provést.

```
public class warrior : hero
{
    private bool using_shield = false;
    Počet odkazů: 1
    public warrior(string name, string[] texture, int health, int armor, int damage, string[] description, string weapon) :
    {
        slots = new string[] { "left", "right", "sword", "shield", "potion" };
    }

    Počet odkazů: 2
    public warrior(warrior w) : base(w)
    {
        slots = new string[] { "left", "right", "sword", "shield", "potion" };
    }
```

Dále má přepsané fukce attack, defend, special, které odpovídjí schopnostem postavy.

```
    public override void attack(enemy e)
    {
        if (stamina >= 5)
        {
            if (position + 1 == e.position)
            {
                stamina -= 5;
                e.TakeDamage(damage);
                LogEvent(name + " zaútočil na " + e.name);
            }
            else
                BadActionMessage = true;
        }
        else
            NotEnoughStaminaMessage = true;
    }

    Počet odkazů: 2
    public override void defend(enemy e)
    {
        if (stamina >= 10)
        {
            stamina -= 10;
            using_shield = true;
            LogEvent(name + " používá štít");
        }
        else
            NotEnoughStaminaMessage = true;
    }

    Počet odkazů: 2
    public override void special(enemy e)
    {
        if (stamina >= 15)
        {
            if (health < max_health)
            {
                stamina -= 15;
                health += 5;
                if (health >= max_health)
                    health = max_health;
                LogEvent(name + " použil lektvar");
            }
            else
                BadActionMessage = true;
        }
        else
            NotEnoughStaminaMessage = true;
    }
```

Upravená funkce pro udělení poškození



```
public override void TakeDamage(int damage)
{
    if (using_shield)
    {
        using_shield = false;
    }
    else
    {
        base.TakeDamage(damage);
    }
}
```

Příklady dalších hero tříd

Další třídy vniklé ze třídy hero, jsou kromě warrior také wizard, archer, bersekr. Jejich kód je v podstatě vždy stejně změněn jako u třídy warrior, například ukázka přepsaných funkcí akce v třídě wizard:

```
public override void attack(enemy e)
{
    if (stamina >= 5)
    {
        if (position + 1 == e.position)
        {
            stamina -= 5;
            e.TakeDamage(damage / 2);
            LogEvent(name + " zaútočil na " + e.name);
        }
        else
            BadActionMessage = true;
    }
    else
        NotEnoughStaminaMessage = true;
}

Počet odkazů: 2
public override void defend(enemy e)
{
    if (stamina >= 10)
    {
        if (position + 3 >= e.position && position + 1 != e.position)
        {
            stamina -= 10;
            e.TakeDamage(damage);
            LogEvent(name + " vyvolal ohnivou kouli");
        }
        else
            BadActionMessage = true;
    }
    else
        NotEnoughStaminaMessage = true;
}

Počet odkazů: 2
public override void special(enemy e)
{
    if (stamina >= 15)
    {
        if (health < max_health)
        {
            stamina -= 15;
            health += 5;
            if (health >= max_health)
                health = max_health;
            e.health -= 5;
            if (e.health <= 0)
                e.alive = false;
            LogEvent(name + " použil prokletí");
        }
        else
            BadActionMessage = true;
    }
    else
        NotEnoughStaminaMessage = true;
}
```

Třída enemy

Je to třída vytvořená z třídy character a je přizpůsobena pro vytvoření nepřátel. Nejdůležitější virtualní funkce je Turn, ta kontroluje hlavně staminu a kolik tahů může nepřítel použít, ve třídách vznikajících je toto přepsané a určuje to chování nepřítele. Obsahuje taky nové protected funkce, které se využívají ve Turn funkci:

- moveLeft – pro posun nepřítele doleva
- moveRight – pro posun nepřítele doprava
- nearAttack – pro útok nablízko
- longAttack – pro útok nadálku
- heal – pro uzdravení

```
public class enemy : character
{
    Počet odkazů: 4
    public enemy(string name, string[] texture, int health, int armor, int damage) : base(name, texture, healt
    {
        position = 6;
    }
    Počet odkazů: 4
    public enemy(enemy e) : base(e)
    {
        position = 6;
    }
    Počet odkazů: 9
    public virtual void Turn(hero player)
    {
        if (stamina <= 0)
        {
            player.regenerateStamina();
            stamina = 20;
            playerTurn = true;
        }
    }
    Počet odkazů: 4
    protected void moveLeft(hero player)
    {
        if (position != player.position + 1 && stamina >= 10)
        {
            position -= 1;
            stamina -= 10;
            LogEvent(name + " se posunul doleva");
        }
        else
            stamina -= 2;
    }
    Počet odkazů: 2
    protected void moveRight(hero player)
    {
        if (position < 6 && stamina >= 10)
        {
            position += 1;
            stamina -= 10;
            LogEvent(name + " se posunul doprava");
        }
        else
            stamina -= 2;
    }
    Počet odkazů: 4
    protected void nearAttack(hero player, int multiply = 1)
    {
        if (player.position + 1 == position && stamina >= 7)
        {
            player.TakeDamage(damage * multiply);
            stamina -= 7;
            LogEvent(name + " zaútočil na " + player.name);
        }
        else
            stamina -= 2;
    }
```

```
    protected void longAttack(hero player, int multiply = 1)
    {
        if (player.position >= position - 3 && stamina >= 10)
        {
            player.TakeDamage(damage * multiply);
            stamina -= 10;
            LogEvent(name + " zaútočil na " + player.name);
        }
        else
            stamina -= 2;
    }
    protected void heal()
    {
        stamina -= 20;
        health += 5;
        if (health >= max_health)
            health = max_health;
        LogEvent(name + " se vyléčil");
    }

}
```

Třída rat

Je vytvořena z třídy enemy, a nejdůležitější je přepis funkce Turn, která využívá již vytvořené funkce pro pohyb a útok, také base pro kontrolu staminy. Další důleživá věc je speciální konstruktor, který jako vstup bere stejnou třídu rat, je to proto, že v projektu mám vytvořené například dvě různé třídy rat a jelikož hráč může vícekrát bojovat i se stejnou postavou, tak využívám již předvytvořené objekty třídy rat a „zkopíruju“ si tak jejich vlastnosti do třídy, která se právě používá pro souboj. Po poražení a výzvy znovu k souboji, tak můžu získat stejnou postavu znovu.

```
public class rat : enemy
{
    public rat(string name, string[] texture, int health, int armor, int damage) : base(name, texture, health, arm
    {
        position = 6;
    }
    public rat(rat r) : base(r)
    {
        position = 6;
    }

    public override void Turn(hero player)
    {
        if (player.position + 1 != position)
        {
            moveLeft(player);
        }
        else
        {
            nearAttack(player);
        }
        base.Turn(player);
    }
}
```

Příklady dalších enemy tříd

Další třídy vniklé ze třídy enemy, jsou kromě rat také skeleton, knight a witch. Jejich kód je v podstatě vždy stejně změněn jako u třídy rat, mění se jenom fukce Turn.

```
public class skeleton : enemy
{
    Počet odkazů: 2
    public skeleton(string name, string[] texture, int health, int armor, int damage) : base(name, texture, health, armor, damage)
    {
        position = 6;
    }
    Počet odkazů: 3
    public skeleton(skeleton s) : base(s)
    {
        position = 6;
    }

    Počet odkazů: 6
    public override void Turn(hero player)
    {
        if (player.position < position - 2)
        {
            moveLeft(player);
        }
        if (player.position == position - 2)
        {
            longAttack(player, 2);
        }
        if (player.position == position - 1)
        {
            if (rand.Next(0, 4) == 0)
                moveRight(player);
            else
                nearAttack(player);
        }
        base.Turn(player);
    }
}
```

```
public class witch : enemy
{
    Počet odkazů: 1
    public witch(string name, string[] texture, int health, int armor, int damage) : base(name, texture, health, armor, damage)
    {
        position = 6;
    }
    Počet odkazů: 1
    public witch(witch w) : base(w)
    {
        position = 6;
    }

    Počet odkazů: 6
    public override void Turn(hero player)
    {
        if (player.position < position - 3)
        {
            moveLeft(player);
        }
        if (player.position >= position - 3 && player.position != position - 1)
        {
            longAttack(player, 2);
        }
        if (health < max_health && rand.Next(0, 5) == 0)
            heal();
        if (player.position == position - 1)
        {
            if (rand.Next(0, 4) == 0)
                moveRight(player);
            else
                nearAttack(player);
        }
        base.Turn(player);
    }
}
```

Třída knight je takový mini boss, takže má navíc schopnost get_armor, která mu vrátí brnění, je to ale omezeno privátní proměnnou wait_until_regen.

```
public class knight : enemy
{
    private int wait_until_regen = 30;
    Počet odkazů: 1
    public knight(string name, string[] texture, int health, int armor, int damage) : base(name, textu
    {
        position = 6;
    }
    Počet odkazů: 1
    public knight(knight k) : base(k)
    {
        position = 6;
    }


    private void get_armor()
    {
        stamina -= 20;
        armor += 2;
        if (armor > max_armor)
            armor = max_armor;
    }


    public override void Turn(hero player)
    {
        wait_until_regen -= 1;
        if (player.position < position - 1)
        {
            moveLeft(player);
        }
        if (player.position >= position - 3 && player.position != position - 1)
        {
            longAttack(player);
        }
        if ((health < max_health || armor < max_armor) && wait_until_regen <= 0)
        {
            wait_until_regen = 30;
            heal();
            get_armor();
        }
        if (player.position == position - 1)
        {
            nearAttack(player, 3);
        }
        base.Turn(player);
    }
}
```

Hlavní program

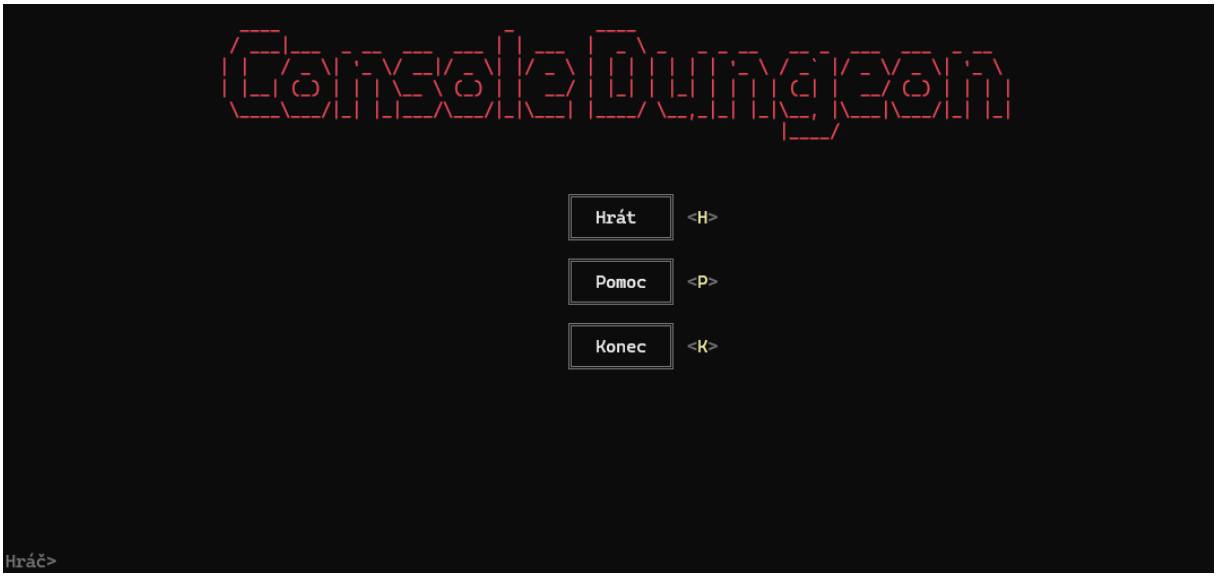
Tady máme vytvoření objektů tříd pro použití.

```
// Objects Setup
hero playedHero;
enemy Enemy;

warrior hero1 = new warrior("Gladiátor", new string[] { "     ___         ", "    |o o|        ", " # _|_-
wizard hero2 = new wizard("Čaroděj", new string[] { "     ___         ", "    |o o|  @@    ", "  _|_-_|_
archer hero3 = new archer("Lukostřelec", new string[] { "    \\\\\\\\\         ", "    |o o|  |\\    ", "
berserk hero4 = new berserk("Bersekr", new string[] { "    .!!!.  ____  ", "    |o o| <__  \\ ", "  _|_-_

rat Rat = new rat("Krysa", new string[] { "                ", "                ", "  OO________    ", "
rat strongRat = new rat("Velka Krysa", new string[] { "                ", "                ", "  OO___
skeleton Skeleton = new skeleton("Kostlivec", new string[] { "        ___     ", "       |x x|    ", "
skeleton BigSkeleton = new skeleton("Kostlivec", new string[] { "         ___    ", "        |x x|    ",
witch Witch = new witch("Čarodějnice", new string[] { "         ___    ", "    @@  |o x|    ", "   || _|_
knight Knight = new knight("Rytíř", new string[] { " _____  ___     ", "|  #__>|0 o|    ", "|_/|| _|<H>|

playedHero = new warrior(hero1);
Enemy = new rat(Rat);
```

Hra běží v nekonečném cyklu a podle globální proměnné „mode“ se určí které menu se má zobrazit. Například funkce MainMenu() vypíše všechen text na obrazovku a vrátí vstup uživatele, který se dále vyhodnotí.

```
// Main Game Loop
while (true)
{
    if (mode == 0) //Main Menu
    {
        option = MainMenu();
        switch (option)
        {
            case "h":
                mode = 2;
                break;
            case "p":
                mode = 1;
                break;
            default:
                HelpMessage = true;
                break;
        }
        if (option == "k")
            break;
    }

    if (mode == 1) // Help Menu
    {
        option = HelpMenu();
        switch (option)
        {
            case "1":
                start_page = 0;
                break;
            case "2":
                start_page = 16;
                break;
            case "3":
                start_page = 16 * 2;
                break;
            case "4":
                start_page = 16 * 3;
                break;
            case "5":
                start_page = 16 * 4;
                break;
            case "z":
                mode = 0;
                break;

            default:
                HelpMessage = true;
                break;
        }
    }
```

Zobrazení MainMenu:



Zobrazení HelpMenu, kde je jednoduché vysvětlení mechanik hry, a taky přesné fungování akcí daných postav (například u třídy warrior, při útoku nablízko se musí nepřítel nacházet vedle hráče):

ChooseHeroMenu slouží pro výběr postavy, vytvoří playedHero jako danou postavu.

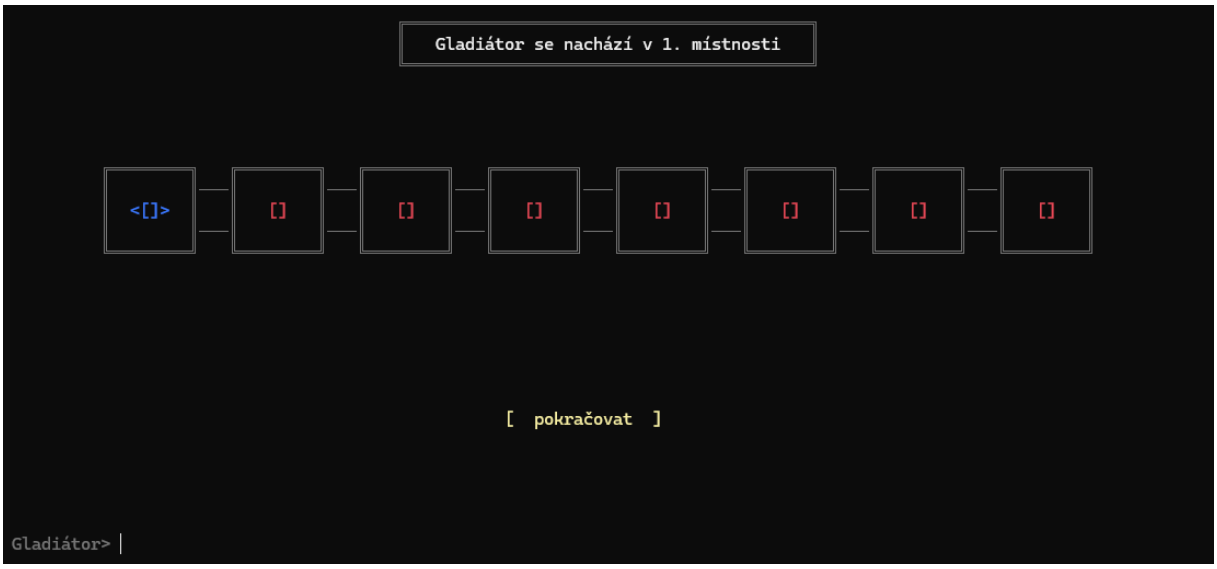
```
if (mode == 2) // Choose Hero
{
    option = ChooseHeroMenu(hero1, hero2, hero3, hero4);
    switch (option)
    {
        case "1":
            playedHero = new warrior(hero1);
            playerName = hero1.name;
            mode = 3;
            break;
        case "2":
            playedHero = new wizard(hero2);
            playerName = hero2.name;
            mode = 3;
            break;
        case "3":
            playedHero = new archer(hero3);
            playerName = hero3.name;
            mode = 3;
            break;
        case "4":
            playedHero = new berserk(hero4);
            playerName = hero4.name;
            mode = 3;
            break;
        case "z":
            mode = 0;
            break;
        default:
            HelpMessage = true;
            break;
    }

}
```

DifficultyMenu je pro nastavení obtížnosti (difficulty – určuje jací nepřátelé se můžou vybrat), dále počet místností (max_levels)

```
if (mode == 3) // Difficulty Menu
{
    option = DifficultyMenu();
    switch (option)
    {
        case "1":
            difficulty = 1;
            max_levels = 3;
            mode = 4;
            break;
        case "2":
            difficulty = 2;
            max_levels = 5;
            mode = 4;
            break;
        case "3":
            difficulty = 3;
            max_levels = 8;
            mode = 4;
            break;
        case "z":
            mode = 0;
            break;
        default:
            HelpMessage = true;
            break;
    }
}
```

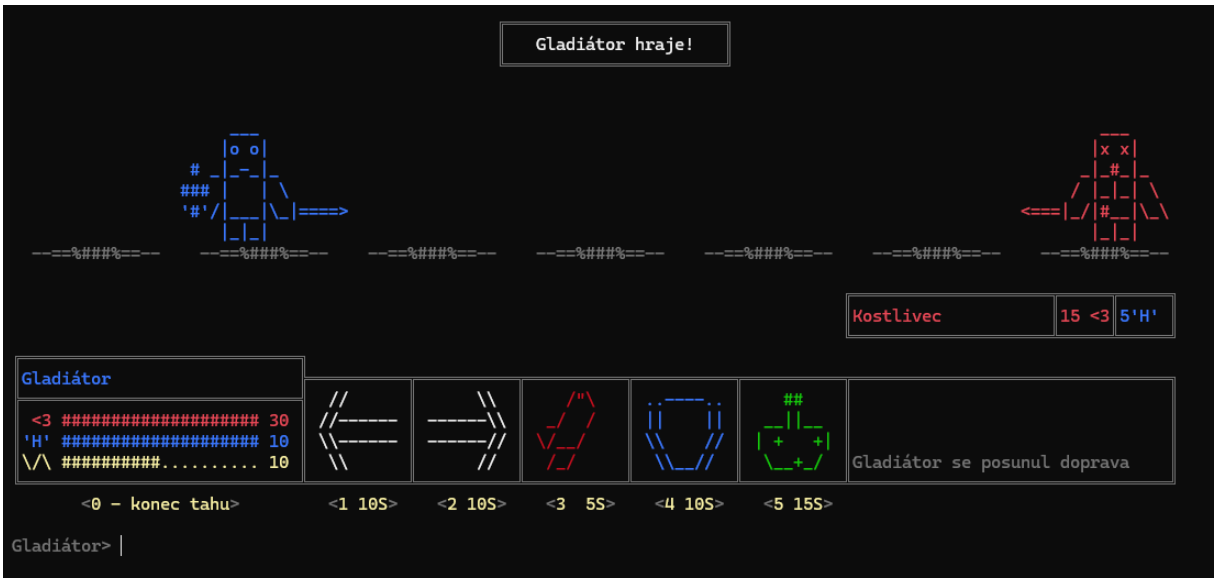
LevelMenu je zobrazení postupu hráče místnostmi a podle obtížnosti vytváří dalšího nepřítele.

```
if (mode == 4) // Level Menu
{
    option = LevelMenu(playedHero);
    if (level < max_levels)
    {
        if (difficulty == 1)
        {
            new_enemy = rand.Next(0, 2);
            if (new_enemy == 0)
                Enemy = new rat(Rat);
            if (new_enemy == 1)
                Enemy = new rat(strongRat);
        }
        if (difficulty == 2)
        {
            new_enemy = rand.Next(0, 5);
            if (new_enemy <= 2)
                Enemy = new skeleton(Skeleton);
            if (new_enemy == 3)
                Enemy = new rat(Rat);
            if (new_enemy == 4)
                Enemy = new rat(strongRat);
        }
        if (difficulty == 3)
        {
            new_enemy = rand.Next(0, 10);
            if (new_enemy <= 3)
                Enemy = new skeleton(Skeleton);
            if (new_enemy == 5 || new_enemy == 6)
                Enemy = new skeleton(BigSkeleton);
            if (new_enemy == 7 || new_enemy == 8)
                Enemy = new witch(Witch);
            if (new_enemy == 9)
                Enemy = new knight(Knight);
        }
        mode = 5;
    }
    if (level == max_levels)
    {
        mode = 6;
    }
}
```



FightMenu slouží pro souboj, kde se hráč i nepřítel mohou pohybovat, střídat se v tahu a podle staminy udělat určité akce. Kromě zobrazení kdo je na tahu, postav na bojišti, se ukazuje také nepřítelovy životy a brnění. Hráči se zobrazuje zdraví, brnění a stamina, dále 5 možných akcí, kde vždy druhá část textu pro vstup určuje kolik staminy daná akce vyžaduje (například pro posun doprava je vstup 2 a vyžaduje 10 staminy) v moment kdy hráči dojde stamina nebo ze strategických důvodů nechce hrát, tak vstupem 0 předá tah nepříteli. Dále úplně nalevo záznam celkem 4 akcí provedených jak hráčem, tak nepřítelem. Taky podle vyhodnocení vítěze boje, se změní proměnná mode, neboli se změní menu, které se zobrazuje, můžeme se vrátit zpátky do LevelMenu nebo přímo do EndMenu.

```
if (mode == 5) // Fight Menu
{
    option = FightMenu(playedHero, Enemy);
    switch (option)
    {
        case "1":
            playedHero.moveLeft();
            break;

        case "2":
            playedHero.moveRight(Enemy);
            break;

        case "3":
            playedHero.attack(Enemy);
            break;

        case "4":
            playedHero.defend(Enemy);
            break;

        case "5":
            playedHero.special(Enemy);
            break;

        case "0":
            playerTurn = false;
            System.Threading.Thread.Sleep(rand.Next(1000, 4000));
            Enemy.Turn(playedHero);
            break;

        default:
            HelpMessage = true;
            break;
    }
    if (!playedHero.alive)
    {
        mode = 6;
    }
    if (!Enemy.alive)
    {
        playedHero.position = 0;
        playedHero.regenerateStamina();
        ClearLogEvent();
        level += 1;
        mode = 4;
    }
}
```



A nakonec menu pro ukončení hry, kde se zobrazí, jestli hráč vyhrál nebo prohrál a dojde k resetu herních proměnných a vrácení do hlavního menu pro možnost spustit hru znovu.

```
if (mode == 6)
{
    option = EndMenu(playedHero);
    mode = 0;
    level = 1;
    playerName = "Hráč";
}
```

Při tvorbě hry jsem si vytvořil svoje vlastní funkce pro zjednodušení vývoje, například v každé menu vrací hodnotu vstupu hráče a já jsem vytvořil vlastní funkci jenom pro zobrazení toho vstupu společně s případným vypsáních chyby, které se hráč dopustil při zadávání vstupu.

```
static string PlayerInput()
{
    if (HelpMessage)
    {
        Console.ForegroundColor = ConsoleColor.Red;
        Console.Write("Špatný vstup - zadejte jednu z možností uvedených v <?>\n");
        HelpMessage = false;
    }
    else
    {
        if (NotEnoughStaminaMessage)
        {
            Console.ForegroundColor = ConsoleColor.Red;
            Console.Write("Nedostatek Staminy!\n");
            NotEnoughStaminaMessage = false;
        }
        else
        {
            if (BadActionMessage)
            {
                Console.ForegroundColor = ConsoleColor.Red;
                Console.Write("Nemožné provést danou akci!\n");
                BadActionMessage = false;
            }
            else
            {
                Console.Write("\n");
            }
        }
    }

    Console.ForegroundColor = ConsoleColor.DarkGray;
    Console.Write(playerName + "> ");
    Console.ForegroundColor = ConsoleColor.White;
    return Console.ReadLine().ToString().ToLower();
}
```

Nebo třeba funkce TextPad pro přidání mezer aby text měl správnou délku a při výpisu zabíral potřebné místo, také určení jesli se má text nacházet na začátku nebo uprostřed.

```
Počet odkazů: 17
static string TextPad(string text, int char_long, bool toMiddle = false)
{
    if (toMiddle)
        return new string(' ', (char_long - text.Length) / 2) + text + new string(' ', char_long - text.Length - (char_long - text.Length) / 2);
    else
        return text.PadRight(char_long);
}
```

Závěr

Vytváření projektu jsem si užil, použití tříd pro vytvoření postav a soubojů není úplně originální nápad, jelikož jsme na něčem podobném učili ve škole právě použití tříd. Myslím, že se to ale právě hodí k takové konzolové hře, při použití na normální hře bych to ale určitě nedoporučil. Každopádně si myslím, že jsem to alespoň originálně zpracoval, hlavně zobrazení každého menu. I s optimalizací kódu, kterou jsem před chvílí zmínil, má program stále 1785 řádků kódu.