kolegij Objektno-orijentirano programiranje, PMFST

UNIVERSITY SIMULATOR

Projektna dokumentacija

Lucija Bročić
lipanj 2018.

Sadržaj

UVOD – UNIVERSITY SIMULATOR

University Simulator je igrica simulacije (simulation video game) u kojoj igrač kao student kroz 10 dana mora proći ispite, tj. poboljšati svoje sposobnosti te do 5.-tog dana doći na 50%, a do zadnjeg, 10.-tog dana na 90% vrijednosti za „knowledge", „charisma" i „fitness", pri čemu su ograničavajući faktori „energy" i „stress level". Igrač se kroz svijet kreće pritiskom na tipke strelice ili slova WASD. Sposobnosti poboljšava u interakciji s objektima u svijet u oko njega, a interakcija se ostvaruje kad igrač dotakne objekt i pritisne tipku „T". Trenutno stanje svojih sposobnosti može vidjeti pritiskom na tipku „Space", a igru može spremiti pritiskom na tipku „Q".

Uz program je priloženo i nekoliko .txt datoteka koje se mogu učitati u igru, sa već spremljenim vrijednostima.

U daljnoj dokumentaciji ću predstaviti kako sam kroz ovaj projekt implementirala osnovne koncepte i ideje objektno orijentiranog programiranja.

PROCEDURALNO I OBJEKTNO-ORIJENTIRANO – PROGRAMIRANJE

Proceduralno programiranje, kao i objektno-orijentirano, je programska paradigma, koja se po svojim karakteristikama razlikuje od objektno-orijentirane paradigme.

Proceduralno programiranje je nastalo iz nestrukturiranog programiranja, koje se koristi za male i jednostavne programe redajući naredbe koje koriste zajednički skup podataka. Potreba za grupiranjem dijelova koda se pojavila jer kad se željelo izvršiti isti kod više puta, trebalo je kopirati kod koliko god puta treba. Izdvajanjem dijelova koda u procedure, program više nije samo slijed naredbi, već slijed procedura. Procedure mogu imati lokalne podatke te ih se može pozivati više puta, rješavajući problem kopiranja naredbi.

Objektno-orijentirano programiranje je proširenje proceduralnog i njegov evolucijski nasljednik. U proceduralnom programiranju, podaci se spremaju u varijable, unaprijed kreirane i imenovane memorijske lokacije u računalu te piše niz naredbi koje manipuliraju tim vrijednostima, često grupirane u procedure koje čine logičke cjeline. U objektno-orijentiranom programiranju, javljaju se objekti koji su konceptualno slični konkretnim objektima u stvarnom svijetu, mogu sadržavati vlastite varijable i metode te atribute. Objektima grupiramo određene podatke i metode u logične cjeline, te ovisno o vrijednosti svih atributa objekta, promatramo različita stanja objekta te njegovo ponašanje.

KLASA I OBJEKT

U osnovi objektno-orijentiranog programiranja leže objekti i interakcija među objektima. Svaki objekt se stvara po određenom obrascu kojeg čini klasa. Dakle, klasa je nacrt u kojem su spremljena različita svojstva i metode, te stvaranjem instance objekta „popunjavamo“ vrijednosti svojstava. Objekt instanciramo pomoću ključne riječi new te konstruktora, public metode u klasi.

Metode su funkcije, poveznica među objektima, objekti vrše interakciju međusobno preko metoda. Za definiranje metode moramo definirati njezino pravo pristupa, vraća li nekakav podatak te njegov tip, ime metode te u zagradama pišemo koje parametre prima. Ako su metode u klasama, to znači da instance te klase mogu koristiti i pozivati te metode.

Primjer konstruktora klase GeneralItem

```
abstract class GeneralItem : Sprite
{
    public GeneralItem(string s, int x, int y, int knowledge, int charisma, int fitness, int stress, int energy)
        :base(s,x,y)
    {
        this.knowledgeChange = knowledge;
        this.charismaChange = charisma;
        this.fitnessChange = fitness;
        this.stressChange = stress;
        this.energyChange = energy;
    }
```

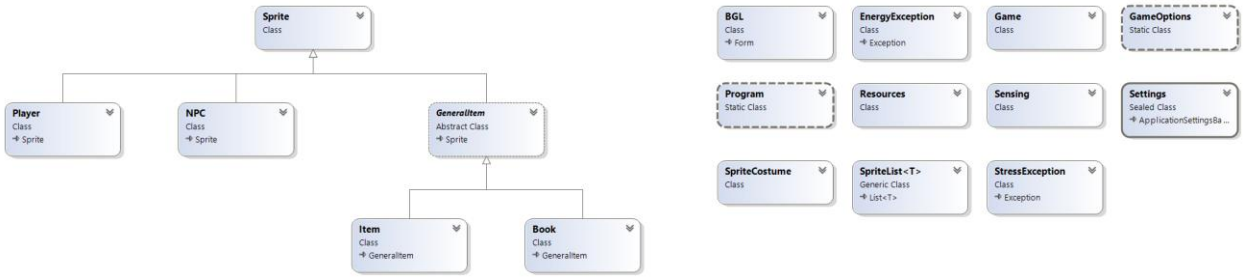
Primjer instanciranja novog objekta klase Player

```
Player player = new Player("sprites\\boy000.png", 100,100);
```

Klase u projektu:

- Apstraktna klasa GeneralItem, te izvedene klase Item i Book
- Klase Player i NPC
- Klase izvedene iz Exception klase – EnergyException i StressException
- Već postojeće klase u OTTER-u (Sprite, Sensing, GameOptions,…)

Dijagram klasa projekta



GeneralItem je apstraktna klasa. Apstraktne klase se ne mogu instancirati, ali možemo izvoditi klase iz njih. Tako ovdje klasa GeneralItem okuplja svojstva koja imaju i Item i Book, primjerice knowledgeChange ili energyChange. Međutim, Item i Book se razlikuju po dvama svojstvima – Item troši vrijeme, a ne novac, dok Book troši novac, ali ne i vrijeme.

Statičke klase su klase koje nije potrebno instancirati da bi se pristupilo njihovim svojstvima i metodama. U projektu je to primjerice klasa GameOptions.

Primjer pozivanja svojstva iz klase GameOptions

```
player.X = 0;
player.Y = GameOptions.DownEdge/2 - player.Heigth;
```

Instance ovih klasa su:

- Item: desk, bed, table, bookcase, tv, karaoke, dumbell, computer, workdesk
- Book: bKnowledge, bCharisma, bFitness, bStress
- NPC: professor, trainer, friend1, friend2
- Player: player

GLAVNE KARAKTERISTIKE OBJEKTNO-ORIJENTIRANOG PROGRAMIRANJA

Glavne karakteristike, tzv. „Četiri stupa objektno orijentiranog-programiranja“ su:

1. Enkapsulacija (učahurivanje)
2. Apstrakcija
3. Nasljeđivanje
4. Polimorfizam

ENKAPSULACIJA

Enkapsulacija je skrivanje i povezivanje unutrašnjih elemenata nekog objekta u povezanu i smislenu cjelinu tako da unutrašnji ustroj nije vidljiv vanjskom svijetu. Često se enkapsulacija očituje ograničavanjem prava pristupa određenim elementima objekta. Primjerice, private polja unutar klase enkapsuliramo public svojstvima te preko metoda pristupa (get i set) unutar tih svojstava pristupamo poljima.

Primjer enkapsulacije:

```
private int knowledge;
public int Knowledge
{
    get { return knowledge; }
    set
    {
        if (value < 0)
            knowledge = 0;
        else if (value > 100)
            knowledge = 100;
        else
            knowledge = value;
    }
}
```

Prava pristupa elementima se mogu definirati prilikom deklaracije korištenjem ključnih riječi:

- public – element vidljiv svima, i izvan klase u kojoj je deklariran
- private – element vidljiv samo unutar klase u kojoj je deklariran
- protected – element vidljiv u klasi u kojoj je deklariran te svim klasama izvedenim iz početne

APSTRAKCIJA

Apstrakcija je vrlo sličan pojam enkapsulaciji. Dok enkapsulacija služi zaštiti podataka od vanjskog svijeta i pogrešnog korištenja, apstrakcija služi skrivanju nepotrebnih dijelova, stvarajući tako

manje kompleksan sustav. U projektu je apstraktna klasa GeneralItem primjer apstrakcije. U nju smo sakrili sve elemente potrebne za klase Item i Book, te tako „očistili“ te klase.

NASLJEĐIVANJE

Postupkom nasljeđivanja možemo iz osnovne klase izvesti podklase. Osnovna klasa je klasa koja služi kao osnova za nasljeđivanje, a izvedena klasa je klasa koja nasljeđuje element osnovne i najčešće sadrži nove.

U projektu je, primjerice, klasa GeneralItem osnovna klasa (iako je i ona podklasa od klase Sprite), a klase Item i Book klase izvedene iz nje.

```
class Item : GeneralItem        class Book:GeneralItem
```

Ako nam neke metode ili svojstva iz osnovne klase nisu prikladni za upotrebu u izvedenoj klasi, onda ih prilagođavamo, ili „premošćujemo.“ Prilikom tog postupka koristimo ključne riječi new i override. Pri korištenju riječi new, koristi se nova implementacija umjesto one u izvedenoj klasi, ali ako instanciramo objekt izvedene klase ako objekt osnovne koristi se svojstvo iz osnovne klase, dok pri korištenju override se poziva posljednje definirana implementacija.

Primjer premošćenog svojstva

```
public override int X
{
    get { return base.X; }
    set {
        if (value < GameOptions.LeftEdge-GameOptions.SpriteWidth/2)
            base.X = GameOptions.RightEdge-GameOptions.SpriteWidth-50;
        else if (value > GameOptions.RightEdge-GameOptions.SpriteWidth)
            base.X = -GameOptions.SpriteWidth / 2;
        else
            base.X = value;
    }
} //override-ano svojstvo za X koordinatu
```

POLIMORFIZAM

Polimorfizam je svojstvo metode da različito djeluje ovisno o kontekstu. Primjerice, metoda SetHeading u klasi Sprite, ako pri pozivu metode pošaljemo parametar tipa DirectionsType, poziva se prva metoda, dok ako pošaljemo parametar tipa int, poziva se druga metoda.

```
/// <summary> Postavlja smjer.
public void SetHeading(DirectionsType heading)...

/// <summary> Metoda koja pomiče lika u 4 osnovna smjera: lijevo, desno, gore i  ...
public void MoveSimple(int steps)...

/// <summary> Postavlja smjer lika.
public void SetHeading(int newDirectionAngle)... //Setdir
```

Slično polimorfizmu se koristi i overload, ili „preopterećenje metode“. To je svojstvo objektno-orijentiranog programiranja koje omogućava postojanje metoda s istim imenom, ali različitim svrhama koje se postižu promjenom broja parametara i/ili promjenom povratnog tipa podataka.

Primjer u projektu je konstruktor klase Item koji može primiti sve potrebne podatke potrebne za interakciju s objektom, dok overload konstruktor može primiti samo podatke bitne za iscrtavanje objekta na zaslon (podatci koje prima svaki Sprite).

```
class Item : GeneralItem
{
    private int timeMin;
    public int TimeMin...

    public Item(string s, int x, int y, int knowledge, int charisma, int fitness, int stress, int energy, int time)...

    public Item(string s, int x, int y) //overload konstruktor...
}
```

IZNIMKE

Iznimka je svako pogrešno ili neočekivano stanje koje se javi prilikom izvršavanja programa. Sve iznimke su objekti klase Exception, te njima upravljamo preko try, catch i finally blokova. U try bloku pišemo kod koji želimo da se izvrši, catch blok hvata iznimke koje se jave prilikom izvršavanja koda u try bloku, a finally blok se koristi za kod koji se svakako mora izvršiti.

U projektu postoje dvije klase izvedene iz klase Exception: StressException i EnergyException.

```
class EnergyException : Exception
{
    private static string message = "You don't have enough energy!";
    public EnergyException()...
}

class StressException: Exception
{
    private static string message = "You are too stressed!";
    public StressException()...
}
```

Hvatanje iznimki u catch bloku

```
if (player.TouchingSprite(desk) && sensing.KeyPressed("T"))
{
    Wait(0.5);

    try
    {
        player.Energy += desk.EnergyChange;
        player.Knowledge += desk.KnowledgeChange;
        player.Stress += desk.StressChange;
        TimerUpdate(desk.TimeMin);
    }
    catch (EnergyException en)
    {
        Wait(0.5);
        MessageBox.Show(en.Message);
    }
    catch (StressException se)
    {
        Wait(0.5);
        MessageBox.Show(se.Message);
    }
```

DOGAĐAJI

Događaj je reakcija na neku pojavu unutar programa. Kao odgovor na događaj definiramo posebne metode za upravljanje događajima („event handler“). Primjer događaja je pritisak tipke na tipkovnici ili mišu. U programu, to bi bili, na primjer, pritisci na tipke za kretanje ili pritisak na dugme na formi.

```
private void buttonStart_Click_1(object sender, EventArgs e)
{
    panelStart.Visible = false;
    panelOpening.Visible = false;

    CreatePlayer();
    SetupGame();
}

private void buttonExitGame_Click(object sender, EventArgs e)...

private void buttonStart_Click(object sender, EventArgs e)...
private void buttonLoadGame_Click(object sender, EventArgs e)...
```

Event handler - metode za upravljanje događajima u klasi Sensing

```
/// <summary> Provjerava je li tipka koja je poslana kao parametar pritisnuta.
public bool KeyPressed(string keyName)
{
    if (KeyPressedTest && Key == keyName)
    {
        Game.WaitMS(20);
        return true;
    }
    else
        return false;
}

/// <summary> Provjerava je li tipka koja je poslana kao parametar pritisnuta.
public bool KeyPressed(Keys key)
{
    if (KeyPressedTest && Key == key.ToString())
    {
        Game.WaitMS(20);
        return true;
    }
    else
        return false;
}
```

DELEGATI

Delegat je objekt koji sadrži referencu na objekt te omogućuje da reference na metodu koristimo kao ulazni argument neke druge metode.

U projektu nisu korišteni delegati, ali primjer deklaracije bi bio:

```
delegate string PrimjerDelegat(int x);
```

Primjerice, taj delegat možemo koristiti za enkapsulaciju bilo koje metode koja za povratni tip ima string, a prima int kao ulazni parametar.

```
public string PrimjerMetoda(int broj)
{
    return broj.ToString();
}
```

SLIKE ZA SPRITE-OVE:

https://opengameart.org/content/character-rpg-sprites

https://opengameart.org/content/lpc-misc-tile-atlas-interior-exterior-trees-bridges-furniture

https://opengameart.org/content/teacher

https://opengameart.org/content/charactertrainer-sprites-64px