Šolski center Novo mesto
Srednja elektro šola in tehniška gimnazija
Šegova ulica 112
8000 Novo mesto

IZDELAVA IGER - TETRIS

(Maturitetna seminarska naloga)

Predmet: Računalništvo

Avtor: Miha Plahuta, T4B
Mentor: Gregor Mede, univ. dipl. inž. rač. in inf.

Novo mesto, april 2022

POVZETEK IN KLJUČNE BESEDE

V maturitetni nalogi sem opisal process izdelave igrice Tetris. V nalogi sem predstavil programski jezik C#, v katerem sem napisal kodo za delovanje igrice ter programsko okolje Microsoft Visual Studio v katerem sem se celotnega projekta lotil. Nato sem v celoti opisal process izdelave iger, kot se tega lotijo velika prodjetja, ki se s tem ukvarjajo vsak dan. Celoten postopek je razdeljen na sedem faz, o katerih sem napisal vse ključne stvari, katere je koristno vedeti, da razumemo celotno zadevo. Za tem sem se lotil Tetrisove zgodovine, načina delovanja, v katerem sem predstavil kako se igrico sploh igra, terminologije, pravil ter predstavil grafične komponente same igrice. Te vključujejo logotip, ki se pojavi na začetnem zaslonu, preden igrico začnemo sploh igrati, ozadje, mrežo ter same like Tetrisa, ki so sestavljeni iz tako imenovanih ploščic. Vso grafiko sem narisal v programu Adobe Illustrator, za katerega sem se odločil zaradi že osvojenega znanja iz preteklosti. Za dodaten korak izdelave svoje grafike sem se odločil zato, da bi igrico res v celoti izdelal sam. Predstavil sem tudi nasvete profesionalnih igralcev Tetrisa, da bi bralcem, ki se odločijo mojo igro preizkusiti, pomagal priti do čim višjega rezultata. Nazadnje sem se lotil še opisa ključnih delov kode, kateri bodo bralcem, ki so malo bolj vešči programiranja, pomagali razumeti delovanje mojega programa.

KLJUČNE BESEDE:

- Programska koda
- Igralna mreža
- Razred
- Metoda
- Igra
- Tetris
- Jezik C#

KAZALO VSEBINE

KAZALO SLIK

1 UVOD

Tetris je tretja najbolj igrana video igra vseh časov, z več kot 100 milijonov kopij prodanih do današnjega dne, ravno s tem namenom pa sem se lotil izdelave le te. Sam sem jo že od otroštva zelo rad igral in mi je še dan danes ena izmed redkih iger, katere se nisem naveličal (1).

Želel sem razumeti celotno logiko za igrico, nato pa jo nekako prilagoditi svojim željam. Moj cilj je bil izdelati lastno igrico, z lastnimi grafičnimi komponentami ter unikatnimi funkcionalnostmi, katere naredijo moj Tetris, boljši ali pa vsaj drugačnejši od ostalih, ki se pojavljajo na trgu.

Prvotno je bil moj namen narediti igrico v programskem okolju Unity, a sem se tik pred začetkom izdelave odločil, da jo bom naredil le v Visual Studiu. Za to sem se odločil, da na nek način dokažem, da se da marsikaj narediti tudi že v samem, preprostem Visual Studiu, saj je dan danes vse več iger na amaterski in profesionalni ravni narejenih v Unity-u. Vedel sem, da bom za to porabil več časa, a sem želel preizusiti tudi sam Visual Studio, saj sem nekaj let nazaj že naredil preprostejšo igrico v Unity-u in sem si zadal izziv, da se je tokrat lotim na nekoliko drugačnejši način.

Dela sem se lotil z raziskovanjem raznih tutorialov, po slovensko vodičev oziroma navodili, s spleta, ki so mi pomagali razumeti kako se lotiti problema ter pokazali nekaj bližnjic za efektivnejše delovanje programa. Izdelava je bila precej lažja, saj se jezika C# skoraj, da nisem rabil od začetka učiti, saj se je izkazal za izredno podobnega jeziku Java, katerega smo v šoli obdelavali veliko časa.

2 IZDELAVA IGER – TETRIS

2.1 PROGRAMSKI JEZIK C#

Programski jezik C# je zasnoval Anders Hejlsberg iz Microsofta leta 2000, pozneje pa ga je kot mednarodni standard odobrila Ecma leta 2002 in ISO leta 2003. Microsoft je predstavil C# skupaj z .NET Framework-om in Visual Studio, oba sta bila zaprtokodna. Takrat Microsoft še ni imel odprtokodnih sistemov. Več kot 10 let pozneje je Microsoft izdal Visual Studio Code (urejevalnik kode), Roslyn (prevajalnik) in poenoteno platformo .NET (programsko ogrodje), ki vsi podpirajo C# in so brezplačni, odprtokodni in medplatformski (2).

Jezik C# se pri skladnji zgleduje po številnih drugih programskih jezikih, najbolj izrazito po C/C++ in javi, kar mi je delo zelo olajšalo, saj smo se jezika Java naučili že v šoli in sem natanko vedel princip delovanja. Jezik je bil skrbno načrtovan z namenom biti čim bolj preprost in moderen ter uporabiti najboljše značilnosti drugih jezikov in popraviti pomanjkljivosti, ki so se pokazale v njih. Omogoča nam bolj optimizirano strojno kodo, večjo varnost pri definiciji tipov (lažje odkrivanje nekonsistentnosti v času prevajanja) in mnogo drugih prednosti (3).

Slika 1: C# logo

(vir: newstore88)

2.2 MICROSOFT VISUAL STUDIO

Microsoft Visual Studio je integrirano razvojno okolje podjetja Microsoft. Uporablja se za razvoj računalniških programov, pa tudi spletnih mest, spletnih aplikacij, spletnih storitev in mobilnih aplikacij. Visual Studio vključuje urejevalnik kode, ki podpira IntelliSense (komponento za dokončanje kode in preoblikovanje kode). Integrirani debugger deluje kot debugger na izvorni ravni in na ravni stroja. Druga vgrajena orodja vključujejo profilator kode, oblikovalec za gradnjo aplikacij GUI, spletni oblikovalec, oblikovalec razredov in načrtovalec shem baze podatkov. Sprejema vtičnike, ki razširijo funkcionalnost na skoraj vseh ravneh (4).

Slika 2: Visual Studio logo

(Vir: blog.hendraptr)

Visual Studio podpira 36 različnih programskih jezikov in omogoča urejevalniku kode in razhroščevalniku, da podpirata (v različni meri) skoraj kateri koli programski jezik, pod pogojem, da obstaja storitev, specifična za jezik. Vgrajeni jeziki vključujejo C, C++, Visual Basic .NET, C#, F#, JavaScript, HTML, CSS ter mnogo ostalih, s katerimi se mi, v srednji šoli, še nismo seznanili (5).

2.3 PROCES IZDELAVE IGER

Video igre so digitalno zasnovane igre, ki se običajno igrajo na osebnih računalnikih ali namenskih igralnih napravah, kot so igralne konzole (npr. Xbox, PlayStation) ali ročne igralne naprave (npr. 3DS, Vita). Igrajo se z interakcijo z uporabniškim vmesnikom ali vhodno napravo in elektronskim manipuliranjem slik, ki jih ustvari računalniški program na monitorju ali drugem zaslonu (6).

Slika 3: Izdelava iger

(Vir: Freepik)

Prva videoigra se je pojavila leta 1952, ko je britanski profesor AS Douglas kot del svoje doktorske disertacije na Univerzi v Cambridgeu ustvaril OXO, znan tudi kot križci in krožci ali tik-tac-toe. Danes videoigre predstavljajo 100 milijard dolarjev vredno svetovno industrijo in skoraj dve tretjini ameriških domov ima člane gospodinjstva, ki redno igrajo video igre. Namenjene so predvsem sproščanju ter zabavi (7).

2.3.1 PLANIRANJE VIDEO IGRE

Tu se začne vsak projekt. Preden programerji začnejo pisati, oblikovalci začnejo oblikovati in razvijalci začnejo razvijati, se mora na plano pojaviti ideja za video igro. To je prvi del faze načrtovanja, iz katerega bo zrasla vsaka video igra (8).

Morda se ne sliši, vendar je zamisel o video igrici eden najtežjih delov razvoja iger. To smo lahko tudi sami občutili pri izbiranju, oziroma odločanju katero igro ali program bomo napisali za končni izdelek pri predmetu računalništvo. Ideja, ki si jo zamislimo ali pa si jo zamisli igralni studio, bo služila kot hrbtenica celotne igre, zato mora biti dobro premišljena. Vprašati se moramo nekaj ključnih vprašanj, za lažje planiranje izdelave igre (9):

- Kakšno vrsto videoiger izdelujemo?
- Bo 2D ali 3D?
- Katere so nekatere ključne lastnosti, ki jih mora imeti?
- Kdo so njeni liki?
- Kdaj in kje poteka?
- Kdo je naša ciljna publika?
- Na kateri platformi to gradimo?

Slika 4: Planiranje igre

(Vir: Vecteezy)

2.3.2 PREDPRODUKCIJA

Naslednja faza razvoja igre razmišlja o tem, kako oživeti številne ideje, predstavljene v fazi načrtovanja. Tukaj programerji, umetniki, oblikovalci, razvijalci, inženirji, vodje projektov in drugi ključni oddelki sodelujejo pri realizaciji vseh idej in ciljev za igro. Ta faza lahko traja od enega tedna do enega leta, odvisno od vrste projekta, virov in razpoložljivih financ ter običajno traja do 20 % celotnega produkcijskega časa.

Umetniki se srečujejo z oblikovalci, da zagotovijo, da so skice, barvne palete in umetniški slogi usklajeni s tem, kar je bilo določeno v fazi načrtovanja. Ti zgodnji vizualni elementi dajejo vizualni vodnik po celotnem videzu in načinu igre (8).

2.3.3 PROIZVODNJA

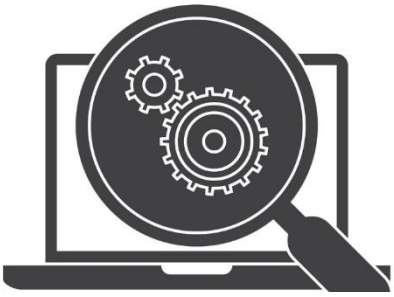
Večina časa, truda in sredstev, porabljenih za razvoj video iger, je v fazi proizvodnje. To je tudi ena najzahtevnejših stopenj razvoja video iger. Med tem postopkom (8).:

- Modeli likov so oblikovani, da so v zgodbi videti točno tako, kot bi morali
- Močno se dela na samih zvočnih efektih v igri
- Ustvarjajo se okolja, ki so dinamična in primerna za različne stile igranja
- Razvijalci napišejo na tisoče vrstic izvorne kode, da oživijo vsak del vsebine
- Vodje projekta določijo mejnike in urnike ter s tem zagotovijo, da celotna ekipa dela tako kot mora. To je še posebej pomembno, če založnik redno preverja stanje napredka v razvoju igre.

2.3.4 TESTIRANJE

Vsako funkcijo in mehaniko v igri je treba preizkusiti za nadzor kakovosti. Igra, ki ni bila temeljito preizkušena, je igra, ki še ni pripravljena za izdajo. Tukaj je nekaj stvari, na katere mora razvijalec biti pozoren (9):

- Ali obstajajo bug-i (hrošči oziroma napake)
- Ali se vse prikazuje na zaslonu?
- Ali vse funkcije zapisane v kodi delujejo
- Ali se v katerem primeru igra “sesuje”



Slika 5: Testiranje
(Vir: Vectorstock

2.3.5 FAZA PRED OBJAVO/IZDAJO IGRE

Faza pred izdajo je stresen čas za igralne studie. Vprašanja dvoma vase se lahko pojavijo, ko se sprašujete, kako se bo javnost odzvala na vaš prvi funkcionalni izdelek. Pojavljajo se še zadnja vprašanja ali igri še kaj manjka, ali je ta produkt zares dovolj dober za na trg. V tej fazi se še naredi tako imenovan napovednik oziroma trailer igre, podobno kot pri filmu, ki pritegne ljudi še preden je igra na voljo za nakup (9).

2.3.6 ZAGON/OBJAVA IGRE

Pred zagonom se sproti še popravi vse napake, ki se naključno pojavijo in igra se objavi. Tukaj se delo celotne ekipe ne konča. Ni nenavadno, da se video igre zaženejo s serijami manjših napak. Prvih nekaj mesecev v fazi po zagonu običajno porabimo za prepoznavanje in odpravljanje teh “hroščev”. Igralni studii se zanašajo tudi na igralce, da oddajo poročila o napakah ali spregovorijo o napakah na spletnih forumih. Vse to je del podpore po zagonu (9).

Drugi del po zagonu je zagotavljanje rednih posodobitev programske opreme za igro. Te posodobitve segajo od popravkov do nove naložljive vsebine ali DLC-jev, za kar sem se odločil, da bom naredil/delal tudi sam. Kadarkoli bom imel viška časa, bom svojemu Tetrisu nekaj dodal. Objavljanje sveže vsebine je običajno v današnji igralniški industriji, ker poveča vrednost in privlačnost igre (10).

Slika 6: Zagon igre
(Vir: Vectorstock)

2.4 VSE O TETRISU

2.4.1 ZGODOVINA

Tako kot mnoge največje zamisli v zgodovini je tudi Tetris nastal povsem nenamerno. Aleksej Pajitnov je bil programski inženir na Sovjetski akademiji znanosti v Moskvi, zadolžen za testiranje novega tipa računalnika, Elektronika 60. S tem namenom je napisal preprosto igro, ki temelji na uganki iz svojega otroštva. Pomagalo bi oceniti, kako zmogljiv je bil računalnik - in zagotovilo malo zabave. Ni pa vedel, da bo nastala igra, postala ena največjih, najbolj zasvajajočih in najuspešnejših iger vseh časov (11).

Pajitnov je črpal navdih iz pentomina, klasične sestavljanke, sestavljene iz vseh različnih oblik, ki jih je mogoče narediti s kombinacijo petih kvadratov (skupaj 12), s ciljem, da jih razporedite v leseno škatlo kot sestavljanko. Igro je poimenoval Tetris, ki združuje grško številko "tetra" (kar pomeni štiri) in tenis, njegov najljubši šport. Pajitnov je bil takoj zasvojen. "Nisem se mogel ustaviti, da ne bi igral te prototipne različice, saj je bilo sestavljanje oblik zelo zasvajajoče," je povedal po telefonu iz Seattla, kjer zdaj živi (12).

Čeprav je Tetris takoj postal priljubljen med programerji z dostopom do Electronike 60, naprava ni imela nobenih grafičnih zmogljivosti in manj pomnilnika kot današnji kalkulatorji. Pod pritiskom zahtev po izdelavi različice igre za IBM PC, bolj razširjen računalnik z boljšo grafiko, je Pajitnov delo dodelil Vadimu Gerasimovu, 16-letnemu študentu, ki je poleti delal v svoji pisarni (danes inženir na Google). Igra se je hitro razširila. "Bilo je kot ogenj na drva. Vsi v Sovjetski zvezi, ki so imeli osebni računalnik, so imeli na sebi Tetris," je dejal Pajitnov (12).

Slika 7: Prvotni Tetris logo
(Vir: Livescience)

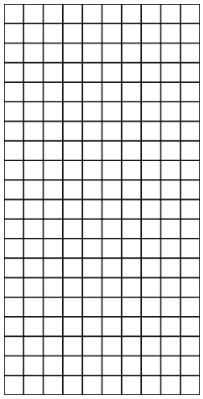
2.4.2 GRAFIČNE KOMPONENTE TETRISA

TERMINOLOGIJA

- **Tetrimino** – igralni block oziroma lik, s katerimi igramo tetris. Imamo 7 različnih likov, v oblikah črk O, I, T, L, J, S, Z.
- **Mino** - eden od štirih kvadratov, ki sestavljajo Tetrimino. Da bi zgradili pravi Tetrimino, se morajo Minos popolnoma povezati na vsaj eni od svojih strani. V moji programski kodi klican “Tile” oz. ploščica
- **Matrix** - to je igralna mreža oziroma prostor, na katerem se igra Tetris. Tetrimini padejo z vrha matrice, nato pa jih igralec obrača in jih premika na želena mesta na dnu.
- **Počisti vrstico** - do tega pride, ko je celotna vrstica zapolnjena z vsemi Mino-ti (10 v vrsti) in s tem se ta vrstica izbriše, vse ostale pa se premaknejo dol.
- **Spusti block** – ko imamo block pripravljen nad mestom, kjer ga želimo imeti, pritisnemo “spusti block”, ki naš lik premakne vse do dol, kolikor gre.
- **Premakni dol** – če nočemo blocka direktno spustiti do dol, ampak le postopoma premikati, klikamo tipko “premakni dol”, ki lik premika le za (-1).
- **Premakni levo/desno** – nam omogoči premikanje lika levo in desno po matriki, z namenom, da ga pripravimo, da pade na željeno mesto.
- **Zavrti** – nam omogoča, da naš lik vrtimo okoli osi, z namenom, da ga nastavimo tako, da se bo prilegal mestu, kamor ga hočemo spustiti. Block lahko vrtimo v levo ali desno smer (13).

IGRALNA MREŽA - MATRIX

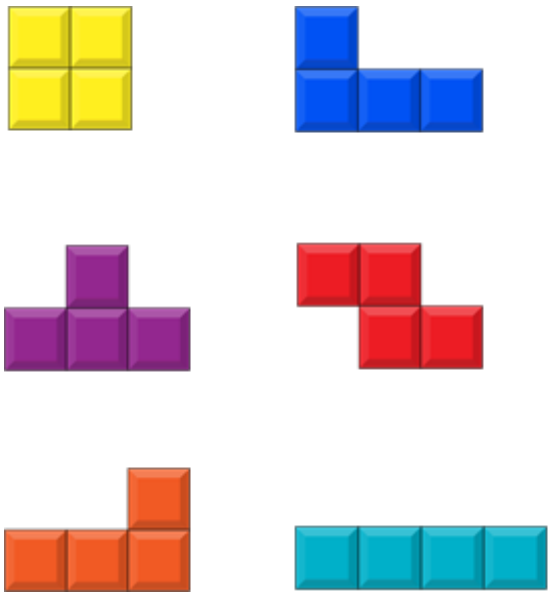
Tetris se igra na mreži velikosti 10x20 (širina, višina), ampak v resnici 10x22, saj sta zgornji dve vrstici namenjeni generiranju/pojavljanju blockov oziroma likov, s tem namenom pa nista vidni igralcu igre. V mojem primeru se igralna mreža generira v programski kodi, zato bom poleg prilepil sliko igralne mreže s spleta (14).



Slika 8: Igralna mreža
(Vir: Deviantart)

IGRALNI LIKI/BLOKI - TETRIMINO

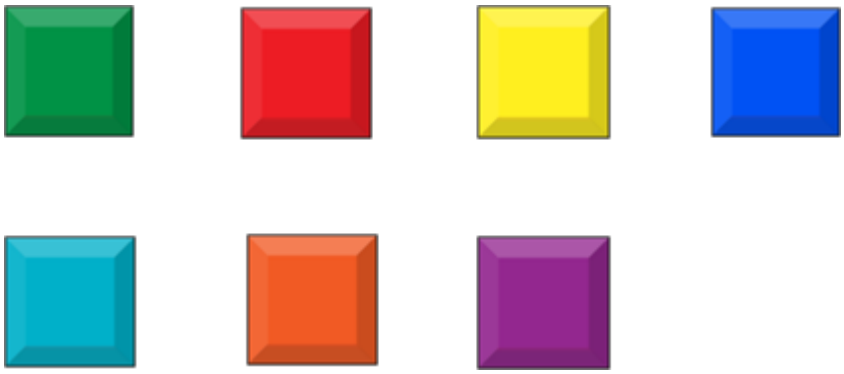
Kot že omenjeno imamo 7 različnih likov, v obliki črk O, I, T, L, J, S, Z. Te lahko rotiramo v levo in desno, premikamo v levo in desno, spuščamo dol, ter samo premikamo dol za eno pozicijo (-1). Vsak block ima svojo barvo, kar popestri igro in nam tudi pomaga pri hitrejši prepoznavi blocka. Vse like sem narisal v Adobe Illustratorju, vključno z ozadjem moje igre (15).



Slika 9: Igralni liki / bloki

KVADRATEK/PLOŠČIČA – MINO – TILE

Iz ploščic so v programu sestavljeni posamezni liki, pa tudi pri obračanju posameznega blocka, se dejansko samo ploščice prestavljajo iz enega mesta na drugega. Nekako so dejansko razporejene v dvodimenzionalni tabelici.



Slika 10: Ploščice / tiles

2.4.3 PRAVILA IGRANJA

Princip igranja je zelo preprost. Namen igre Tetris je doseči čim večje število točk, to pa dosežemo s čiščenjem vrstic. Vrstica se počisti takrat, ko jo v celoti zapolnimo s ploščicami različnih likov. Ko se vrstica počisti se vse ostale vrstice premaknejo dol in s tem dobimo več prostora za postavljanje novih. Da to lažje naredimo imamo razne funkcije, katere nam omogočajo premikanje, vrtenje in spuščanje likov. Te bom podrobneje opisal v praktičnem delu naloge, s slikami programske kode in natančnim opisom le te (16).

Da ni igra preveč preprosta, se s številom počiščenih vrstic pospešuje tudi premikanje blockov. To nam čez čas izjemno uteži in postane postavljanje blockov skoraj da nemogoče pri ogromnih velikostih in posledično nam ne uspe več slediti igri in pravočasno postaviti vse blocke. Ko se nam igralna mreža zapolni do vrha smo igro izgubili. Prikažejo se nam dosežene točke, koliko vrstic smo očistili in do katerega levela smo prišli. Če smo presegli naše rekordno število točk, se nov rekord zabeleži v datoteko kjer imam shranjen moj program (17).

Slika 11: Pravila (Vir: Flaticon)

2.4.4 TOČKOVANJE

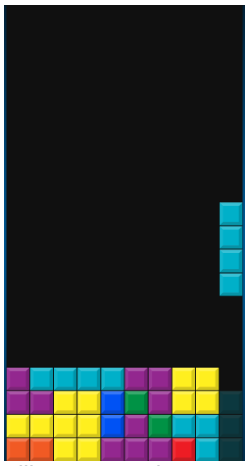
Za Tetris je značilen poseben način točkovanja. Več vrstic kot počistiš na enkrat, več točk dobiš. Na ta način lahko z enakim številom počiščenih vrstic pridemo do več točk, kot če bi čistili samo eno po eno vrstico. V tabeli so prikazane točke pridobljene z različnim številom počiščenih vrstic v eni potezi, torej s postavitvijo le enega blocka (običajno je to I-Block, oziroma lik v obliki črke i). Število pridobljenih točk se povečuje tudi z višanjem levela, oziroma stopnje. Število pridobljenih točk se zmnoži z levelom, saj višji kot smo level, težje je dosegati točke, saj igra stalno postaja hitrejša (16).

Table 1: Točkovanje

Level	Točke- 1 vrstica	Točke- 2 vrstici	Točke- 3 vrstice	Točke– 4 vrstice
1	40	100	300	1200
2	80	200	600	2400
3	120	300	900	3600
...				
n	40 * (level)	100 * (level)	300 * (level)	1200 * (level)

2.4.5 TEHNIKE IGRANJA

Igra Tetris obstaja že zelo dolgo in tako kot pri vseh ostalih stvareh, ljudje poskušamo biti, kar se da dobri v njej. S tem namenom so ljudje skozi igranje te igre, iznašli načine, s katerimi najlažje, oziroma najhitreje dosežemo veliko število točk in s tem premagamo našega nasprotnika. Kot že prej omenjeno, nam počiščenje štirih vrstic na enkrat prinese največ točk in sicer 1200. Temu se reče tudi Tetris. Spodaj navedenih je nekaj nasvetov, katere je za navdušene igralce Tetrisa podal sedemkratni svetovni prvak v Tetrisu Jonas Neubauer (18).



Slika 12: Igranje ravno

“PLAYING FLAT” oziroma igranje ravno, je strategija, katero Neubauer priporoča vsem začetnikom, ki želijo narediti prvo stvar v smeri izboljšave svojega igranja. To je način zlaganja kosov na način, ki bo ustvaril, kar se da ravno zgornjo vrsto kosov. Ravno igralno

polje pomeni, da imate več možnosti za odlaganje kosov. Splošno pravilo: nikoli ne zlagajte kosov več kot dva bloka visoko in ne ustvarjajte luknje, ki je več kot dva bloka globoka. Nemogoče je zapolniti luknjo, ki je tri ali več globoka z ničemer, razen s kosom v obliki črke i. Zlaganje blokov z višino treh ali več prav tako poveča težave pri izravnavanju (19).

RAZUMITE ROTACIJSKI SISTEM blockov

SPREJEMAJTE HITRE ODLOČITVE pri rotaciji in postavitvi blocka

SPREMLJAJTE NASLEDNJI BLOCK, ki bo prišel za zdajšnjim

“VELIKO VADITE” je Jonasov zadnji nasvet, ki velja za več ali manj vsako stvar, ki se jo želimo naučiti ali pa postati mojster v njej. Zavedamo se, da je tudi na osnovni ravni veliko stvari, ki jih je treba upoštevati od začetka. Tudi ob predani praksi bo trajalo nekaj časa, da te ideje postanejo druga narava. Sčasoma boste odkrili edinstvene nastavitve in lastne trike za boj proti težkim situacijam (19).

2.5 PREDSTAVITEV IZDELAVE IGRE/KODE

Sedaj, ko smo se seznanili s teorijo same izdelave iger, principi delovanja Tetrisa ter nekaj nasveti, pa bom še predstavil izdelavo igre Tetris še z vidika programerja. Opisal bom le nekaj najbolj ključnih delov kode, ki se največkrat izvedejo ali pa brez njih preprosto ne bi šlo. Celoten program je bil napisan v programskem okolju Visual Studio 2022, zato bodo tudi vse slike za lažjo predstavo pridobljene iz lastnega dokumenta oziroma projekta odprtega v Visual Studiu.

2.5.1 RAZRED IGRALNA MREŽA

V tem delu bom predstavil razred IgralnaMreza, kjer se nahajajo vse metode, ki manipulirajo z vsemi elementi na igralni površini. Prva stvar, ki sem jo naredil je igralna mreža. Za izdelavo le te uporabimo preprost ukaz, katerega smo vajeni že iz Jave. Uporabil sem stavek `private readonly int[,] grid;`. S tem sem ustvaril dvodimenzionalno tabelo, katera bo služila kot igralna mreža, po kateri se bojo premikali liki. Za tem sem ustvaril več metod:

`AliJeZnotrajMreze`

Metoda, katera bo kasneje uporabljena pri postavljanju blockov, preveri pa nam ali sta vrstica in stolpec kamor hočemo premakniti block prazna, torej, da ne gremo izven igralne mape. Preveri torej ali je x-os igralne mreže večja od 0, ali je manjša od širina naše igralne mreže, katero smo mi nastavili na 10 in ali je koordinata y od blocka znotraj igralne površine zgoraj in spodaj, torej manjša od 22 in večja od 0.

```
public bool AliJeZnotrajMreze(int r, int c)
{
    return r >= 0 && r < Vrstice && c >= 0 && c < Stolpci;
}
```

Slika 13: Metoda AliJeZnotrajMreze

`AliJeCelicaPrazna`

Podobno kot prejšnja metoda deluje tudi metoda `AliJeCelicaPrazna.` Ta nam, kot že ime pove, preveri ali je celica oziroma ploščica (tile) prazna, saj mora naš lik, ki je sestavljen iz 4 ploščic, se prilegati na mesto, kamor ga hočemo postaviti.

```
public bool AliJeCelicaPrazna(int r, int c)
{
    return AliJeZnotrajMreze(r, c) && grid[r, c] == 0;
}
```

Slika 14: Metoda AliJeCelicaPrazna

`AliJeVrsticaDokoncana` in **`AliJeVrsticaNedokoncana`**

V programu pogosto uporabljena metoda je tudi `AliJeVrsticaDokoncana`, ki preveri ali je vrstica že dokončana in metoda `AliJeVrsticaNedokoncana`, ki stori ravno obratno. Obe metodi vsebujeta zanki, ki se premikata po vrstici in preverjata ali je prostor za ploščico blocka, katerega želimo tam postaviti.

```
public bool AliJeVrsticaDokoncana(int r)
{
    for (int c = 0; c < Stolpci; c++)
    {
        if (grid[r, c] == 0)
        {
            return false;
        }
    }
    return true;
}
```

Slika 15: Metoda AliJeVrsticaDokoncana

```
public bool AliJeVrsticaNedokoncana(int r)
{
    for (int c = 0; c < Stolpci; c++)
    {
        if (grid[r, c] != 0)
        {
            return false;
        }
    }
    return true;
}
```

Slika 16: Metoda AliJeVrsticaNedokoncana

`IzbrisiVrstico` se prav tako premika po vrstici, torej po y-ih in enega in po enega izbriše, oziroma njegovo vrednost v tabeli nastavi na 0. Ta metoda se uporabi takrat, kadar celo vrstico zapolnimo z blocki in se mora ta izbrisati.

```
private void IzbrisiVrstico(int r)
{
    for (int c = 0; c < Stolpci; c++)
    {
        grid[r, c] = 0;
    }
}
```

Slika 17: Metoda IzbrisiVrstico

`PremakniVrsticoDol` se izvede za metodo `IzbrisiVrstico` in vrstice premakne po igralni mreži dol, kolikor je možno. To, kot vidimo stori tako, da se ponovno z zanko for premika po vrstici in ploščice eno po eno premika dol za toliko, kot smo izbrisali vrstic.

```
private void PremakniVrsticoDol(int r, int SteviloVrstic)
{
    for (int c = 0; c < Stolpci; c++)
    {
        grid[r + SteviloVrstic, c] = grid[r, c];
        grid[r, c] = 0;
    }
}
```

Slika 18: Metoda PremakniVrsticoDol

Metoda **IzbrisiPolneVrstice** je sestavljena iz nekaj prej omenjenih metod in z njimi najprej preveri ali je vrstica dokončana, nato pa če je, kliče metodo `IzbrisiVrstico` in s tem izbriše zapolnjeno vrstico ter poveča spremenljivko “izbrisane” za 1, kar pomeni da se je število izbrisanih vrstic povečalo za 1. Nato preveri še, če je število izbrisanih vrstic večje kot 0, torej če smo sploh katero izbrisali in če smo, kliče metodo `PremakniVrsticoDol` s pomočjo katere premakne vrstice za toliko mest, kot je bilo preštetih izbrisanih vrstic.

```
public int IzbrisiPolneVrstice()
{
    int izbrisane = 0;

    for (int r = Vrstice-1; r >= 0; r--)
    {
        if (AliJeVrsticaDokoncana(r))
        {
            IzbrisiVrstico(r);
            izbrisane++;
        }
        else if (izbrisane > 0)
        {
            PremakniVrsticoDol(r, izbrisane);
        }
    }
    return izbrisane;
}
```

Slika 19: Metoda IzbrisiPolneVrstice

2.5.2 RAZRED GAME STATE / STANJE IGRE

V tem razredu se elementi moje igre začnejo premikati. Opisal bom le nekaj ključnih premikov, saj bi sicer porabil ogromno prostora, da bi vse opisal v sami seminarski, ker je igra Tetris precej kompleksna in vsebuje kar nekaj metod, da deluje tako kot mora in ima vse funkcionalnosti originalnega Tetrisa. V tem razredu določimo velikost igralne mreže, generirajo se nam tako imenovani čakalni blocki, ki so liki, kateri se nam prikazujejo med igranjem igrice na levi strain in prikazujejo kateri je naslednji block, ki se bo pojavil za tem, ko postavimo na željeno mesto tistega, katerega imamo “v roki”. Prav tako ustvarim nekaj spremenljivk, katere se skozi igranje igrice spremnijajo na različne vrednosti. Vse te so podrobneje opisane v sami kodi, katera bo bila v prilogi.

Metoda **`GameState`**, kot že omenjeno, pomeni stanje igre. V tej metodi se zgodijo ključne spremembe stanja igre. Ustvari se igralna mreža, pripravi in zamenja se tudi čakalni block, level se nastavi na 1 ter spremenljivka “zacetniMeni” se nastavi na true, saj je bil to edini način, na katerega sem lahko nastavil, da se igra ni začela takoj ko smo VS project zagnali, ampak se je namesto tega, najprej pojavil začetni meni, ki vsebuje logo mojega Tetrisa in napis, ki nas opozori, da moramo za začetek igre pritisniti tipko Enter.

```
public GameState(bool zacetniMeni = true)
{
    IgralnaMreza = new IgralnaMreza(22, 10);
    CakalniBlock = new CakalniBlock();
    CurrentBlock = CakalniBlock.ZamenjajNaslednjiBlock();
    ZačetniMenu = zacetniMeni;
    Level = 1;
}
```

Slika 20: Metoda GameState

`PrileganjeBlocka`

Med vsemi napisanimi metodami se najpogosteje uporabi metoda PrileganjeBlocka, ki preveri ali se bo block, katerega "držimo v roki" prilegal mestu, nad katerim ga držimo. To stori tako, da za vsako koordinato posamezne ploščice, iz katerih je sestavljen posamezen lik, preveri ali bi se prilegala poziciji na dnu, preveri pa tudi če slučajno sega čez rob mreže ali katerega drugega lika. Ta metoda bo uporabljena tudi vedno, ko bomo želeli naš block zavrteti, saj se pri vrtenju blockov pogosto zgodi, da če smo preblizu stene, oziroma roba naše igralne površine, da bi lik segal čez rob.

```
private bool PrileganjeBlocka()
{
    foreach (Pozicija p in CurrentBlock.TilePozicije())
    {
        if (!IgralnaMreza.AliJeCelicaPrazna(p.Vrstica, p.Stolpec))
        {
            return false;
        }
    }
    return true;
}
```

Slika 21: Metoda PrileganjeBlocka

`ZavrtiBlockLevo`

Sedaj bom opisal le eno izmed metod, ki omogočajo vrtenje naših likov okoli svoje osi, premikanje levo in desno ter spuščanje dol. Vse so narejene na skoraj enak način, zato ni potrebno opisovanje vsake posebej. Metoda `ZavrtBlockLevo` pokliče metodo `ZavrtiLevo()`, ki je v razredu "Block" in zavrti block v levo.

```
public void ZvartiBlockLevo()
{
    CurrentBlock.ZavrtiLevo();

    if (!PrileganjeBlocka())
    {
        CurrentBlock.ZavrtiDesno();
    }
}
```

Slika 22: Metoda ZavrtiBlockLevo

V tem delu kode računamo število točk. To storimo z inicializacijo spremenljivke SteviloTock, ki se nastavi na število, katerega prejme iz metode IzbrisiPolneVrstice, katero smo poklicali iz razreda IgralnaMreza, vrne pa nam za vsakim izbrisom, število počiščenih vrstic, na podlagi tega števila, pa s switch zanko izračunamo in povečamo število doseženih točk. To storimo preprosto tako, da v primeru da je bili v tistem koraku počiščenih npr. 3 vrstice, potem se spremenljivka Score oziroma točke, poveča za 40*Level v katerem smo ta trenutek, poveča pa se še spremenljivka PocisceneVrstice, ki nam stalno šteje koliko smo vse skupaj počistili vrstic. Na podlagi te številke pa se sproti tudi povečuje level. Za vsakih 10 vrstic, ki smo jih počistili se nam level poviša za 1.

```
int SteviloTock = IgralnaMreza.IzbrisiPolneVrstice();
switch (SteviloTock)
{
    case 1: Score += 40*(Level); PocisceneVrstice += 1; break;
    case 2: Score += 100*(Level); PocisceneVrstice += 2; break;
    case 3: Score += 300*(Level); PocisceneVrstice += 3; break;
    case 4: Score += 1200*(Level); PocisceneVrstice += 4; break;
}
Level = PocisceneVrstice / 10 + 1;
```

Slika 23: Switch stavek

2.5.3 RAZRED MAIN WINDOW / GLAVNO OKNO

Razred glavno okno je to, kar že ime pove. Vsa koda v tem oknu določa to, kar bo igralec mojega Tetrisa videl. V tem razredu so glavnejše metode, ki služijo predvsem prikazovanju component na zaslon in izvajajo samo igro.

`Draw`

Metoda `Draw` se prav tako uporablja precej pogosto in sicer na začetku vsake igre, ko se vsi elementi in komponente igrice tako rekoč “narišejo” na zaslon. To so “Grid” oziroma igralna mreža, “GhostBlock” je block, ki je enak tistemu, katerega držimo, le da se na igralni površini nariše najbolj na dnu, kolikor gre, nariše se z manjšo vidnostjo, kar pa nam omogoča predogled kam bo naš lik priletel, ko ga spustimo do dol, ali pa postopoma pade do dol. Metoda nam prav tako izpiše/izpisuje trenutno število točk, level, ter število počiščenih vrstic v tej igri. Z ukazom string readText pa sem iz datoteke, shranjene na mojem USB ključku, prebral do sedaj najvišje število doseženih točk, ki se posodobi za vsako, ko to število presežemo.

```
private void Draw(GameState gameState)
{
    DrawGrid(gameState.IgralnaMreza);
    DrawGhostBlock(gameState.CurrentBlock);
    DrawBlock(gameState.CurrentBlock);
    DrawNextBlock(gameState.CakalniBlock);
    ScoreText.Text = $"Točke: {gameState.Score}";
    LevelText.Text = $"Level: {gameState.Level}";
    PocisceneText.Text = $"Pociscene: {gameState.PocisceneVrstice}";

    string readText = File.ReadAllText("G:\\Tetris\\Tetris\\REZULTATI.txt");
    RekordText.Text = $"Rekord: {Int32.Parse(readText)}";
}
```

Slika 24: Metoda Draw

ZaustaviIgro

Metoda ZaustaviIgro se izvede šele, ko igro končamo ročno, torej s klikom tipke ESC in klikom na gumb, ki konča igro ali pa, ko se igra konča avtomatsko, ker nam ni uspelo več sproti postavljati blocke in brisati vrstice. Prav tako nastavi "GameOverMenu" na vidno ter "GamePauseMenu" na skrito, v primeru, da smo ga prej imeli odprtega, zaradi zamena ročne zaustavitve igre. Zadnja stvar, ki jo ta metoda preveri je to, kar sem že prej omenil in sicer ali smo morda presegli rekord in če smo ga, številko shranjeno na datoteki posodobi na sedaj dosežene točke.

```
public void ZaustaviIgro()
{
    GameOverMenu.Visibility = Visibility.Visible;
    GamePauseMenu.Visibility = Visibility.Hidden;
    FinalScoreText.Text = $"Točke: {gameState.Score}";
    FinalLevelText.Text = $"Level: {gameState.Level}";
    FinalPodrtihText.Text = $"Počiščene vrstice: {gameState.PocisceneVrstice}";

    string readText = File.ReadAllText("G:\\Tetris\\Tetris\\REZULTATI.txt");
    if (Int32.Parse(readText) < gameState.Score)
    {
        File.WriteAllText("G:\\Tetris\\Tetris\\REZULTATI.txt", gameState.Score.ToString());
    }
}
```

Slika 25: Metoda ZaustaviIgro

GameLoop

Najbolj pomembna metoda celotnega programa pa je GameLoop, ki je zanka, ki se izvaja neprestano, dokler ne velja kateri izmed pogojev: KonecIgre je enak true, UstavitevIgre je enak true ali pa je ZačetniMenu enak false. V zanki se ob vsakem počiščenju vrstice poveča hitrost premikanja blockov navzdol, oziroma delay med vsakim premikom blocka se zmanjša. Za vsako se tudi block avtomatsko premika navzdol in pa seveda je klicana metoda `Draw(gameState)`, ki nam prikazovanje vsake spremembe sploh omogoči. Kot prej omenjeno pa, če se zgodi UstavitevIgre, katero izvedemo mi, s klikom na ESC, pa se GamePauseMenu pojavi, tako da mu nastavimo Visibility na visible. Prav tako prikažemo FinalScoreText in FinalLevelText, ki prikažeta dosežene točke in level.

```
private async Task GameLoop()
{
    Draw(gameState);
    while (!gameState.KonecIgre && !gameState.UstavitevIgre && gameState.ZačetniMenu)
    {

        int delay = (maxDelay - (gameState.PocisceneVrstice * delayDecrease));
        await Task.Delay(delay);
        gameState.PremakniBlockDol();
        Draw(gameState);
    }

    if (gameState.UstavitevIgre)
    {
       GamePauseMenu.Visibility = Visibility.Visible;
        FinalScoreText.Text = $"Točke: {gameState.Score}";
        FinalLevelText.Text = $"Level: {gameState.Level}";
        FinalPodrtihText.Text = $"Počiščene vrstice: {gameState.PocisceneVrstice}";

    }
    else if (!gameState.ZačetniMenu)
    {

    }
    else
    {
        ZaustaviIgro();
    }
}
```

Slika 26: Metoda GameLoop

`Window_KeyDown`

Koda, ki nam omogoča igranje igre s pritiskanjem tipk na tipkovnici in miški, pa je zapisana z metodo `Window_KeyDown`. V tej metodi imamo switch stavek, ki zaznava pritiske različnih tipk, ti pa izvedejo posamezne funkcije oziroma metode, katere smo naredili v razredu gameState. Ker sem skoraj vse te že predstavil, ne bom še tukaj vsakega posebej predstavljal, ampak bom le dal primer, da si lahko ne-programerski bralci moje seminarske naloge predstavljajo kako to izgleda.

Na primer, če kliknemo tipko A, se izvede metoda `ZavrtiBlockLevo`, ki kot sem že prej razložil, v nekaj korakih zavrti naš lik v levo smer. Prav tako sem že omenil, da ob kliku tipke ESC, se izvede metoda `JeIgraUstavljena`, ki našo igro ustavi.

```
private async void Window_KeyDown(object sender, KeyEventArgs e)
{
    switch (e.Key)
    {
        case Key.Left:
            gameState.PremakniBlockLevo();
            break;
        case Key.Right:
            gameState.PremakniBlockDesno();
            break;
        case Key.Down:
            gameState.PremakniBlockDol();
            break;
        case Key.D:
            gameState.ZvartiBlockDesno();
            break;
        case Key.A:
            gameState.ZvartiBlockLevo();
            break;
        case Key.Space:
            gameState.DropBlock();
            break;

        case Key.Escape:
            gameState.JeIgraUstavljena();
            break;

        case Key.Enter:
            gameState.ZačetekIgre();
            GameStartMenu.Visibility = Visibility.Hidden;
            await GameLoop();
            break;

        default:
            return;
    }

    Draw(gameState);
}
```

Slika 27: Metoda Window_KeyDown

Funkcija, ki pa v tem delu kode izstopa, pa je, ko pritisnemo tipko Enter. Izvede se metoda `ZačetekIgre`, ki spremeni spremenljivko, ustvarjeno v gameState-u, imenovano ZačetniMenu, nastavi pa jo na true. Ta funkcija je uporabljena le na začetku, ko igro šele zaženemo in je prikazan začetni menu imenovan GameStartMenu. S pritiskom tipke Enter se vidljivost tega menuja nastavi na skrito in zažene se zanka GameLoop o kateri sem prej govoril, da je odgovorna za celoten potek moje igre Tetris.

`ZacniPonovnoKLIK`

Za konec opisovanja pomembnejših ali pa pogosto uporabljenih metod, pa bom predstavil še metodo, ki nam omogoča uporabo tipk v igrici. Predstavil bom metodo `ZacniPonovnoKLIK`, ki nam omogoča ponovni začetek igre, v primeru, da se je igra zaključila, ker nismo morali več slediti postavljanju likov na igralno površino dovolj hitro.

```
private async void ZacniPonovnoKLIK(object sender, RoutedEventArgs e)
{
    gameState = new GameState();
    GameOverMenu.Visibility = Visibility.Hidden;
    await GameLoop();
}
```

Slika 28: Metoda ZacniPonovnoKLIK

3 ZAKLJUČEK

Pri izdelavi maturitetne seminarske naloge sem se naučil zelo veliko. Dosegel sem svoj cilj, izdelati funkcionalno igro Tetris, uspelo pa mi je mnogo več kot le to. Poleg tega sem se naučil tudi ogromno o procesih izdelave video iger velikih podjetij, kako se pravilno lotiti problema, ki se pojavi med pisanjem kode in tako dalje. Izvedel sem o funkcijah in bližnjicah programskega okolja Microsoft Visual Studio, za katere prej sploh nisem vedel, da obstajajo. Raziskal sem celotno logiko in zgodbo za nastankom ene izmed najbolj znanih iger na svetu.

Zaradi dobrega načrta tega, kako se bom lotil izdelave igre in seminarske naloge, mi je uspelo narediti vse, kar sem si zadal, kar pa ne pomeni, da sem tu končal. Igra ima zelo velik potencial ravno zato, ker imam v mislih še mnogo idej, katere bi moj Tetris naredile zares unikaten.

Menim, da je bila odločitev, da se izdelave, ne ravno preproste igre, lotim v samem Visual Studiu pravilna. Dokazal sem sam sebi in drugim, da je to popolnoma možno narediti, brez kakršnegakoli drugega programskega okolja kot je npr. Unity. Z igrico sem presenetil tudi prijatelje, je pa vprašanje kakšen bi bil produkt, če bi se izdelave lotil s pomočjo Unity-ja.

Možnosti nadaljnega razvoja so ogromne in tudi idej imam obilico, nekaterih se že zdaj sproti lotevam. Vidim možnost, da bi jo enkrat v prihodnosti tudi objavil in omogočil igranje le te, tudi širši javnosti. S projektom sem zelo zadovoljen in ga s ponosom predstavim komurkoli zainteresiranem.

4 ZAHVALA

Zahvaljujem se vsem, ki so kakorkoli prispevali k izdelavi ali pa že samo h motivaciji pri izdelavi moje seminarske naloge. Še posebej pa bi se rad iskreno zahvalil mentorjema Gregorju Medetu univ. dipl. inž. rač. in inf. ter dr. Albertu Zorku univ. dipl. inž. el., ki sta me naučila toliko o programiranju skozi vsa 4 leta ter sta bila pripravljena svetovati glede kakršnegakoli programerskega problema.

Za konec bi se rad zahvalil še svoji družini za vso podporo in vzpodbudo ter bratrancu, na katerega sem se lahko zanesel, ko nisem imel več idej za reševanje programskega problema.

5 VIRI IN LITERATURA

1. **Gaedke, . dailyinfographic**. *What are the Best Selling Video Games of All Time?* [Online] 14 Januar 2022. [Cited: 5 Marec 2022.] https://dailyinfographic.com/what-are-the-best-selling-video-games-of-all-time.

2. **Wikipedia. *C Sharp (programming language).*** [Online] 11 April 2022. [Cited: 5 Marec 2022.] https://en.wikipedia.org/wiki/C_Sharp_(programming_language).

3. **Bill Venners, Bruce Eckel. Artima. *Generics in C#, Java, and C++.*** [Online] 26 Januar 2004. [Cited: 5 Marec 2022.] https://www.artima.com/articles/generics-in-c-java-and-c#part2.

4. **Wikipedia. *Microsoft Visual Studio*.** [Online] 12 April 2022. [Cited: 5 Marec 2022.] https://en.wikipedia.org/wiki/Microsoft_Visual_Studio.

5. **Davis, . quick-adviser**. *How many languages we can use in Visual Studio?* [Online] 28 Oktober 2020. [Cited: 5 Marec 2022.] https://quick-adviser.com/how-many-languages-we-can-use-in-visual-studio/.

6. **igi-global**. *What is Video Games.* [Online] [Cited: 5 Marec 2022.] https://www.igi-global.com/dictionary/video-games/31544.

7. ***Video Game History.*** [Online] History, 10 Junij 2019. [Cited: 5 Marec 2022.] https://www.history.com/topics/inventions/history-of-video-games.

8. **Pickell, . g2. *The 7 Stages of Game Development*.** [Online] 15 Oktober 2019. [Cited: 5 Marec 2022.] https://www.g2.com/articles/stages-of-game-development.

9. **Stefyn, . cgspectrum. *How video games are made**: the game development process.* [Online] 10 Oktober 2019. [Cited: 5 Marec 2022.] https://www.cgspectrum.com/blog/game-development-process.

10. **gulfbusiness.** *Revoland becomes the first blockchain game to launch on Huawei Cloud.* [Online] 7 April 2022. [Cited: 5 Marec 2022.] https://gulfbusiness.com/revoland-becomes-the-first-blockchain-game-to-launch-on-huawei-cloud/.

11. **Wikipedia.** ***Jonas Neubauer****.* [Online] 12 Marec 2022. [Cited: 5 Marec 2022.] https://en.wikipedia.org/wiki/Jonas_Neubauer.

12. **Prisco, . edition.cnn**. *Tetris: The Soviet 'mind game' that took over the world.* [Online] 1 November 2019. [Cited: 2022 Marec 5.] https://edition.cnn.com/style/article/tetris-video-game-history/index.html.

13. **Miranda, . tetris.** ***Tetris Lingo Every Player Should Know****.* [Online] 16 Junij 2017. [Cited: 12 Marec 2022.] https://tetris.com/article/35/tetris-lingo-every-player-should-know.

14. **tetris.fandom.** ***Playfield****.* [Online] https://tetris.fandom.com/wiki/Playfield.

15. **tetris.fandom.** ***List of rules (as of 2009).*** [Online] https://tetris.fandom.com/wiki/Tetris_Guideline.

16. **tetris.wiki.** ***Scoring****.* [Online] 5 Februar 2022. [Cited: 12 Marec 2022.] https://tetris.wiki/Scoring.

17. **codewars.** ***Tetris Series #1 — Scoring System.*** [Online] 2022. [Cited: 12 Marec 2022.] https://www.codewars.com/kata/5da9af1142d7910001815d32.

18. **Dam, Rikke Friis. interaction-design.** ***A Game Explained*** *(an example of a single game and how it meets the rules of fun).* [Online] 18 Junij 2021. [Cited: 12 Marec 2022.] https://www.interaction-design.org/literature/article/a-game-explained-an-example-of-a-single-game-and-how-it-meets-the-rules-of-fun.

19. **Ramos, . Polygon.** ***Tetris tips from a seven-time world champion****.* [Online] 22 Februar 2019. [Cited: 12 Marec 2022.] https://www.polygon.com/guides/2019/2/22/18225349/tetris-strategy-tips-how-to-jonas-neubauer.

6 STVARNO KAZALO

7 PRILOGE

Priloga 1 – Povezava do PowerPoint dokumenta

https://github.com/plahutamihaa/TETRIS2.0/tree/main/PP_Dokument

Priloga 2 – povezava do Word dokumenta

https://github.com/plahutamihaa/TETRIS2.0/tree/main/Word_Dokument

Priloga 3 – povezava do programske kode

https://github.com/plahutamihaa/TETRIS2.0/tree/main/TETRIS_Dokument