ŠOLSKI CENTER VELENJE
ELEKTRO IN RAČUNALNIŠKA ŠOLA
Trg mladosti 3, 3320 Velenje

MLADI RAZISKOVALCI ZA RAZVOJ SAŠA REGIJE

RAZISKOVALNA NALOGA

ZLORABA DELOVNEGA POMNILNIKA PRI RAČUNALNIŠKIH IGRAH

Tematsko področje: RAČUNALNIŠTVO

Avtorja:
Mark Kotnik
Valentin Ozimic

Mentor:
Islam Mušić

Velenje, 2023/2024

Raziskovalna naloga je bila opravljena na Elektro in računalniški šoli Velenje, 2024.

Mentor: Islam Mušić

Datum predstavitve:

KLJUČNA DOKUMENTACIJSKA INFORMACIJA

ŠD Elektro in računalniška šola Velenje, šolsko leto 2023/2024

KG aplikacija / programiranje / goljufanje / delovni pomnilnik / igre

AV KOTNIK, Mark / OZIMIC, Valentin

SA MUŠIĆ, Islam

KZ 3320 Velenje, Trg mladosti 3

ZA ŠC Velenje, Elektro in računalniška šola, 2024

LI 2024

IN Zloraba delovnega pomnilnika pri računalniških igrah

TD Raziskovalna naloga

OP

IJ SL

JI SL / EN

AI Igranje računalniških iger je popularna aktivnost, še posebej med mladimi. Okoli tega je nastala cela skupnost entuziastičnih igralcev, ki v tako imenovanih “e-športih” tekmujejo tudi za denar in druge nagrade. Tako kot pri drugih računalniških področjih, je elektronske naprave možno tudi zlorabljati, to pa nekateri igralci računalniških iger naredijo, da lahko pri igranju goljufajo. V tej raziskovalni nalogi sva se poglobila v goljufanje v računalniških igrah z zlorabo vrednosti v računalniškem delovnem pomnilniku. Zanimalo naju je branje pomnilnika, iskanje pomnilniških naslovov in vpliv teh podatkov na zmožnost goljufanja ter koriščenje pridobljenih podatkov. Naloga je pokazala, da lahko z branjem in spreminjanjem vrednosti v pomnilniku vplivamo na rezultat meta kocke. Pokazala sva tudi, da je od kode in uporabljene tehnologije programa odvisno, če lahko v aplikaciji na tak način goljufamo ter da je možno brati tudi podatke drugih igralcev.

KEY WORDS DOCUMENTATION

ND Elektro in računalniška šola Velenje, šolsko leto 2023/2024

CX aplikacija / programiranje / goljufanje / delovni pomnilnik / igre

AU KOTNIK, Mark / OZIMIC, Valentin

AA MUŠIĆ, Islam

PP 3320 Velenje, Trg mladosti 3

PB ŠC Velenje, Elektro in računalniška šola, 2024

PY 2024

TI Abusing computer memory of computer games

DT Raziskovalna naloga

NO

LA SL

AL SL / EN

AB Playing computer games is becoming an increasingly popular activity, especially among the youth. A whole community of enthusiastic gamers, who compete in so-called "e-sports" for money and other prizes has formed around it. Like in other computer-related fields, electronic devices can also be abused, which some players of computer games do in order to cheat while playing. In this research project, we delved into cheating in computer games through memory reading and manipulation. We were interested in memory reading, searching for game memory addresses, the impact of code on cheating capabilities, and the utilization of acquired data. The study showed that by reading and modifying values in memory, we can influence the outcome of dice rolls, that it depends on the code of the program whether cheating in the application can be done in this way, and that it is possible to read the data of other players.

Kazalo

Kazalo slik

Slovar

Cheat – program, ki bere in spreminja pomnilnik v uporabnikovo korist

Angl. – angleški izraz

Singleplayer – vrsta igre, v kateri je samo 1 igralec

Multiplayer – vrsta igre, ki omogoča igranje z več igralci

Igralni ban – onemogočenost igranja igre, blokiranje dostopa do igre s strani ustvarjalca za določen ali nedoločen čas

E-šport - igranje iger za različne nagrade, kjer v veliki večini igrajo profesionalne ekipe

Streaming platforma – spletna platforma, na kateri lahko pretočno predvajamo video vsebine

ESP – angl. extra sensory perception – uporaba programa, ki igralcu pusti dostop do nasprotnikove lokacije, omogoča, da vidi skozi stene in podobno.

Anticheat – program, ki je vključen v igro, ki poskuša preprečiti goljufanje s preverjanjem sumljivih procesov na igralčevem računalniku in z zaznavo sprememb datotek igre

Pomnilnik igre – prostor, kjer se shranijo vsi podatki igre in vse, kar se v igri dogaja

Zunanji program – program, ki z dostopom do pomnilnika igre bere in spreminja različne informacije

Notranji program – program, ki spremeni kodo igre in omogoči goljufanje

Pointer – kazalnik na naslov v pomnilniku

Boti – umetna inteligenca, ki predstavlja igralca, ki pa je sestavljen iz algoritmov

Online – tehnologija ali dejavnost, ki poteka s prenosom podatkov prek interneta

UI – angl. user interface – uporabniški vmesnik, ki omogoča interakcijo z aplikacijo in ji daje nek izgled. V primeru igre izpisuje na ekran nekih statistik, kot so življenske točke igralca.

1 HIPOTEZE

1. Z branjem in spreminjanjem delovnega pomnilnika lahko vplivamo na igro Met kocke.
2. Tehnologija, ki je uporabljena pri razvoju igre, vpliva na našo zmožnost zlorabe delovnega pomnilnika.
3. V igri lahko beremo podatke nasprotnikov.

2 PREGLED OBJAV

2.1 ZGODOVINA RAZVOJA IGER IN GOLJUFANJA

2.1.1 Čisti začetek razvoja

Prve oblike videoiger so se začele leta 1958 z razvojem igre "Tennis for Two". Igre so postale velika uspešnica, na splošno so se igre hitro razvijale in dosegle arkadni nivo (retro igre) v letu 1971.

Starejše igre so bile krajše in zelo ponavljajoče, saj v tistem času ni bilo orodij za boljše metode ustvarjanja iger. Igre so bile enostavne, imele so tudi enostaven princip. Večinoma so bile kontrolirane z ročko ali kakšno drugo vrsto preprostega kontrolerja.

2.1.2 Kode za goljufanje

Ustvarjalci iger so za testiranje uporabljali kode za goljufanje (angl. cheat code). Te niso bile namenjene vsem igralcem. Ko so igralci odkrili te kode, so jih začeli uporabljati. Primer učinka the kod:

- Neomejeno število življenj
- Preskoči stopnjo igre
- Neomejeno število denarja
- Večja hitrost igralca

Kode za goljufanje so igro naredile bolj zabavno in ji dodale občutek skrivnostnosti. Širile so se skozi različne revije ali pa od ust do ust.

Ustvarjalci iger so videli, da so igralci pozitivno odreagirali in od takrat naprej začeli objavljati takšne kode bolj javno. Naredile so igro bolj zabavno in dale možnost igranja na več načinov.

2.1.3 Dosežki in problemi z goljufanjem

Leta 1980 so se v igrah prvič pojavili dosežki, ki so igralcem nudili možnost osvojitve digitalnih trofej in jim dali občutek, da so nekaj dosegli. Ta dodatek se je hitro širil in prevzel svet igranja iger. Tu so se začeli tudi prvi problemi glede uporabe takšnih kod, saj so nekateri igralci dosežke pridobili na pravilen in zaslužen način, medtem ko so jih drugi pridobivali z uporabo kode za goljufanje.

Istega leta so ustvarjalci dodali tudi možnost, da lahko v igri vidimo dosežke drugih igralcev in se z njimi primerjamo. Ustvarili so lestvico najboljših, kjer so se zabeležila imena in dosežene točke različnih igralcev. Od tu naprej so ustvarjalci začeli iz igre odstranjevati posebne kode.

2.1.4 Razvoj iger in načinov goljufanja

Z letom 1990 in napredkom tehnologije so se nadgradile tudi igre. Razvijalci so začeli posvečati več pozornosti zgodbi in likom, kar je prineslo številne priljubljene naslove z zapletenimi zgodbami in raznolikimi liki. Igre so prišle na domače računalnike in bile dostopne vsem.

Istočasno je razvoj večigralskih iger dosegel vrhunec s širjenjem interneta. Spletno igranje z drugimi igralci v realnem času je postalo glavna značilnost mnogih iger, od strelskih do športnih.

S prihodom pametnih telefonov in tablic so se igre preselile tudi na mobilne naprave, kar je povečalo njihovo dostopnost. Igre so postale vsakodnevna aktivnost za nekatere, saj igralci lahko uživajo v njih kjerkoli in kadarkoli.

Napredek v umetni inteligenci je prinesel bolj inteligentne nasprotnike in soigralce, kar igralcem omogoča bolj dinamično izkušnjo. V zadnjih letih se je pojavila tudi virtualna resničnost, ki igralcem omogoča popolnoma drug občutek igranja iger (in gledanja).

Danes imamo igre z odlično grafiko, globokimi zgodbami, realistično fiziko in raznovrstnimi možnostmi igranja. Enoigralske izkušnje nadaljujejo razvijanje poglobljenih zgodb, medtem ko večigralsko igranje prinaša socialni vidik in tekmovanje v realnem času. Raznolikost žanrov in tehnološki napredek nenehno oblikujeta nove videoigre.

S takšnim uspešnim razvojem iger, so se razvijali tudi načini za goljufanje v igrah. Programerji programov za goljufanje so vedno našli nove načine, kako premagati zaščito igre in v njej goljufati. Skozi čas pa je goljufanje postajalo tudi bolj in bolj popularno zaradi večje želje igralcev po zmagi.

Igre so začele dodajati različne podprograme, ki so deloma brali, kaj igralec počne, imenovan "anticheat" in nato se odločili, če goljufa ali ne. Tisti, ki so goljufali so v veliki večini za svoja dejanja bili kaznovani s prepovedjo igranja. [1]

2.2 VPLIV GOLJUFANJA NA E-ŠPORTNIH DOGODKIH

"E-šport" se je začel okoli leta 1980. Igrali so igre, ki so bile osredotočene na čas, ki ga tekmovalec porabi, da opravi oz. zaključi igro. Skozi čas so se takšni načini spremenili v tekmovalne igre, pri katerih je bilo možno igrati v večigralskem načinu, v katerem so se ekipe med seboj spopadle. Med najbolj popularnimi igrami, ki so se na e-šport dogodkih igrale, so bile strelske igre. Seveda se na začetku takšnih dogodkov ljudje niso zbirali, temveč so igrali na daljavo in gledalci so imeli možnost ogleda iger preko "streaming platform". V začetku e-športa ta zadeva ni bila popularna.

Na začetku leta 2000 so se mnenja začela spreminjati in e-šport je postal velika uspešnica. Posledično so e-športi postali še večji dogodki z občinstvom in pretočnim predvajanjem v živo.

Popularnost zelo hitro narašča, kar pomeni, da se hkrati povečuje število igralcev. Med igralci imamo tudi skupine, ki na različne načine goljufajo. Goljufanje na velikih tekmovanjih je redkost, saj igralce ves čas spremlja večja skupina ljudi. Za takšne primere uporabljajo različne programe, ki na skrivaj igrajo igro namesto nas v ozadju.

Primeri programov, ki so uporabljeni za goljufanje na tekmovanjih:

- Program, ki na nasprotnika nameri s tem da simulera človeške premike miške. Kakršna koli vrsta esp goljufanja (uporaba programa, ki igralcu pusti dostop do nasprotnikove lokacije, omogoča da vidi skozi stene in podobno) tukaj ni možna.

Ob tem se seveda lahko vprašamo, kako lahko ljudi, ki uporabljajo takšen program sploh izpostavimo. To naredimo z uporabo “anticheata. To je program, ki je vključen v igro in poskuša preprečiti goljufanje s preverjanjem sumljivih procesov na igralčevem računalniku in z zaznavo sprememb datotek igre.

Drugi način je seveda, da gledalci to izpostavijo. Ob tem administratorji dogodka pregledajo računalnik igralca, kar se je na profesionalni sceni že zgodilo.

2.3 NAČINI ZA GOLJUFANJE

2.3.1 Programi

Goljufanje je možno z uporabo različnih programov. Te programe v grobem delimo na dve vrsti in sicer na zunanje (angl. external) in notranje (angl. internal) skupine programov.

2.3.2 Notranji programi

So programi, ki dostopajo do igre na preprost način, s tem da v igro dodajajo svojo kodo z uporabo .dll knjižnic. To je bolj tvegan način, saj je znan po tem da igra igralca velikokrat kaznuje s prepovedjo igranja. Če pa je ta programsko dobro napisan je lahko glede na funkcije in uporabnost močnejši in boljši od zunanji programov, saj lahko dodajo svoje funkcije. Večinoma so takšni programi lažje zaznani s strani "anticheata" in uporabnika kaznujejo za uporabo.

2.3.3 Zunanji programi

So programi, ki so težji za izvedbo, ampak zagotavljajo boljšo zaščito pred "anticheati". V nekaterih primerih tudi takšno, ki jo je nemogoče zaznati. Povežejo se na pomnilniku igre, iz katerega lahko berejo ali vanj pišejo. Bolj varno je, da samo berejo, saj to ne sproži kakršnega nevarnega signala igri. Slabost takšnih programov je, da lahko spreminjamo in dodajamo samo stvari, ki jih igra že ima v svoji kodi.

2.3.4 Drugi načini goljufanja

Ustvarjalci takšnjih programov pa so postali tudi zelo kreativni glede na izvedbo. Ugotovili so, da lahko goljufamo s pomočjo čisto drugega računalnika ali mikrokrmilnika (arduino, raspberry PI), ki sta zmožna povezave na glavni računalnik. Z njimi lahko procesiramo in beremo informacije, a ne moremo pisati v pomnilnik. To je zelo dober način za goljufanje, brez da bi igra to zaznala, a trenutno ni tako aktualen, ker so drugi načini lažji. Primer uporabe je, da nam drug računalnik ali mikrokrmilnik simulira premike miške. [2] [3] [4]

2.4 PREPREČEVANJE GOLJUFANJA

Preprečevanje goljufanja pri video igrah je bitka, ki se je začela že pred dolgimi leti, saj igralci nenehno iščejo nove načine goljufanja. Razvijalci iger nenehno popravljajo in izboljšujejo svoje igre, zlasti na področju varnosti. Sistemi proti goljufijam vključujejo avtomatske preglede, ki zaznavajo sumljive dejavnosti, kot so hitre spremembe v zmogljivosti ali nenavadno obnašanje igralcev. Na voljo so tudi algoritmični mehanizmi, ki prepoznajo nepravilnosti v igralnih vzorcih in opozarjajo na morebitne goljufije.

2.5 PROGRAMI, KI PREPREČUJEJO GOLJUFANJE

2.5.1 Valve Anticheat

Valve Anticheat program je razvilo podjetje Valve, ki je na začetku imelo načrte zaščititi njihovo igro. Njihova zaščita deluje na algoritmih, ki zaznavajo gibe v igri, ki jih človeški igralec ni zmožen narediti. Program nima notranjega dostopa oz. notranjih podatkov, kar pomeni, da ga enostavno premagaš. Njegova šibka točka je ta, da ne more zaznati branja podatkov, a lahko zazna pisanje v pomnilnik. [5]

2.5.2 Riot Vanguard

Vanguard program je novejša oblika zaščite, ki ima notranji dostop do celotnega računalnika igralca. Deluje tako, da konstantno pregleduje podatke in zgodovino igralca. Njegovo zaščito lahko zaobidemo z uporabo različnih gonilnikov. Razvilo ga je podjetje Riot Games in je uporabljen v igri Valorant. [6]

2.5.3 EAC

Je plačljiva zaščita, ki jo lahko kupi vsak razvijalec iger. Deluje podobno kot Valve Anticheat, a je bolj priljubljena. Programerji so našli veliko različnih metod, da se ji izognejo. [7]

2.6 AVTORITATIVNI STREŽNIK ZA ONLINE IGRE

2.6.1 Namen

Avtotaritivni strežniki (angl. authoritative servers) so strežniki, namenjeni nadzoru iger. V online igrah, kjer skupaj igra več igralcev, bi goljufanje le enega od teh uničilo izkušnjo drugim igralcem, zato je tovrstne igre potrebno zaščititi proti goljufanju, kar lahko naredimo tudi s tem, da pomembne podatke zapisujemo in beremo s strežnika.

2.6.2 Delovanje

Med strežnikom in odjemalcem (igro) se ves čas pošiljajo podatki. Ko se igralec s strani odjemalca npr. premakne, se zahteva za to pošlje strežniku, ki jo preveri in izvede ter pošlje rezultate nazaj odjemalcu.

To sicer povzroči zamik med pritiskom gumba in prikazom na zaslonu. To lahko preprečimo s predikcijo vrednosti, tako da ob pritisku posodobimo odjemalca in pošljemo zahtevo strežniku. Ob prevelikem odstopanju, kar se bo najverjetneje zgodilo v primeru goljufanja, se odjemalcu pošlje popravek.

Čar tega je, da drugi igralci vidijo igralca, ki goljufa in se npr. teleportira naokoli tam, kjer se dejansko nahaja in ne na poziciji, ki si jo je ta nastavil prek spreminjanja vrednosti v pomnilniku. Avtoritativni strežnik igre večinoma uporabljajo v kombinaciji z anticheati. [8] [9]

3 MATERIALI IN METODE

3.1 IZDELAVA PROGRAMOV

Glavna metoda raziskovanja je izdelava programov, s katerimi lahko potrdiva ali ovrževa zastavljene hipoteze ter izdelava igre, s katero lahko testirava delovanje programov za goljufanje pri določenih pogojih in načinih zapisa izvorne kode igre.

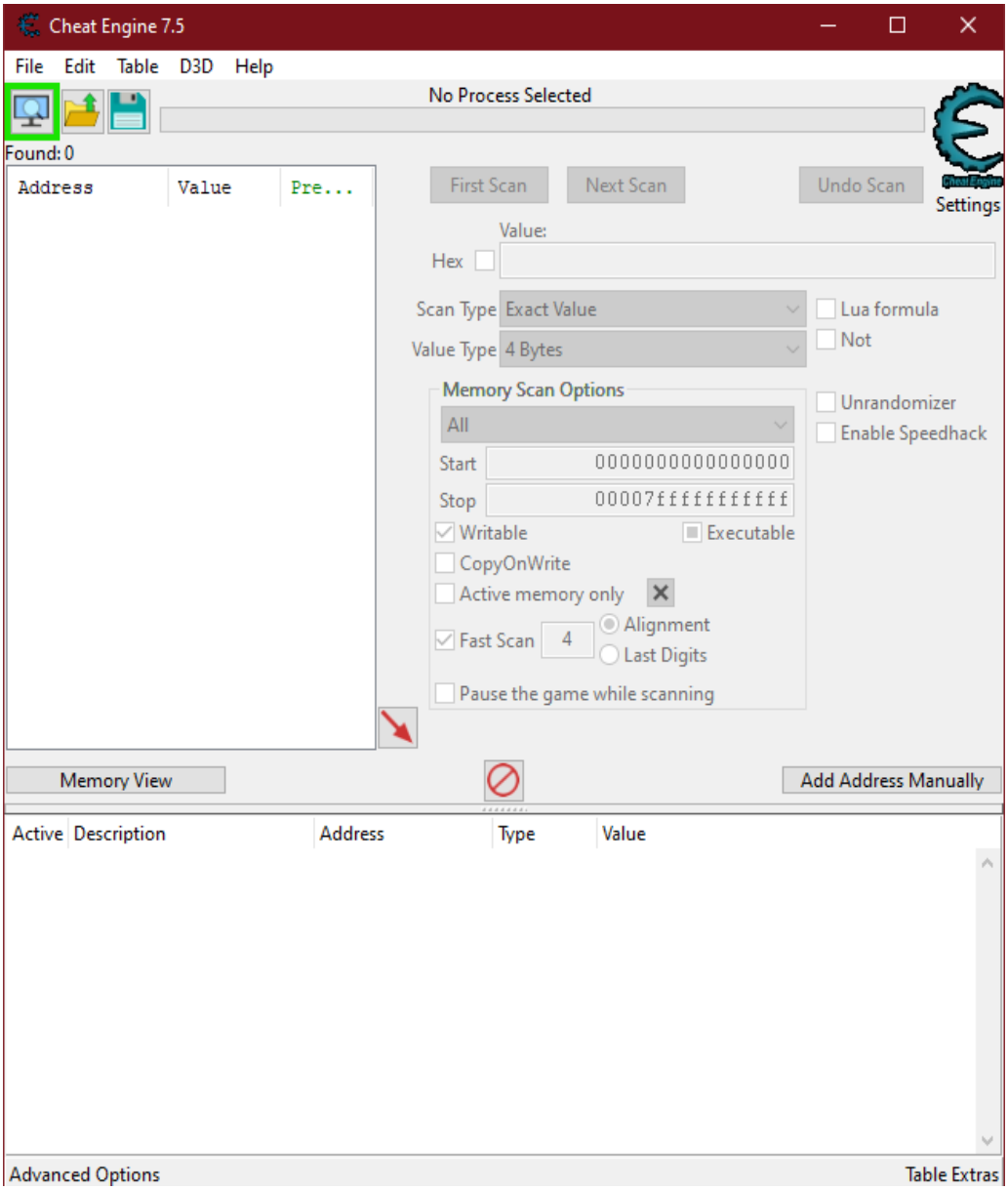
3.2 PROGRAMI, ORODJA, MODULI IN KNJIŽNICE

3.2.1 Cheat Engine

Za to nalogo sva se odločila za uporabo orodja Cheat Engine. Program omogoča branje in spreminjanje pomnilnika iger ter aplikacij. Ta način deluje samo pri tistih igrah oz. aplikacijah, ki niso povezane ali sinhronizirane s strežnikom oz. podatki, ki jih želimo spreminjati, niso sinhronizirana, torej ne delujejo na večini večigralskih iger. Je tudi odprtokodno in ima še več drugih možnosti, kot so:

- Spreminjanje vrednosti
- Ogledovalnik pomnilnika
- Hitrostno goljufanje

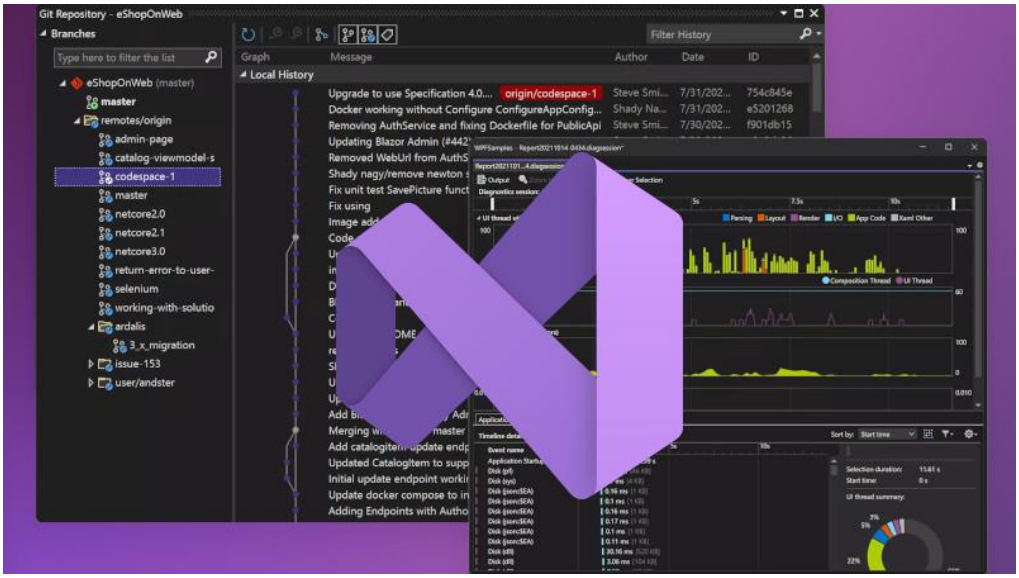
Program ima tudi funkcijo iskanja v delovnem pomnilniku, ki v pomnilniku najde vse vrednosti povezane z igro oz. aplikacijo, ki imajo zapisano vrednost, katero iščemo. S tem orodjem lahko enostavno najdemo določene pomnilniške naslove, kot so življenjske točke, število preostalega streliva, virtualne valute oz. denar in podobno. [10]



Slika 1: Izgled Cheat Engine orodja (vir: lasten)

3.2.2 Visual Studio

Visual Studio je integrirano razvojno okolje, ki omogoča tudi programiranje aplikaacij za operacijski sistem Windows s pomočjo ogrodja Forms. Je odlična izbira za najin projekt, ker je uporaba preprosta in hitra, podpira pa tudi knjižnici memory.dll in swed.dll (opisani spodaj), s katerima lahko beremo in spreminjamo vrednosti v pomnilniku. [11]



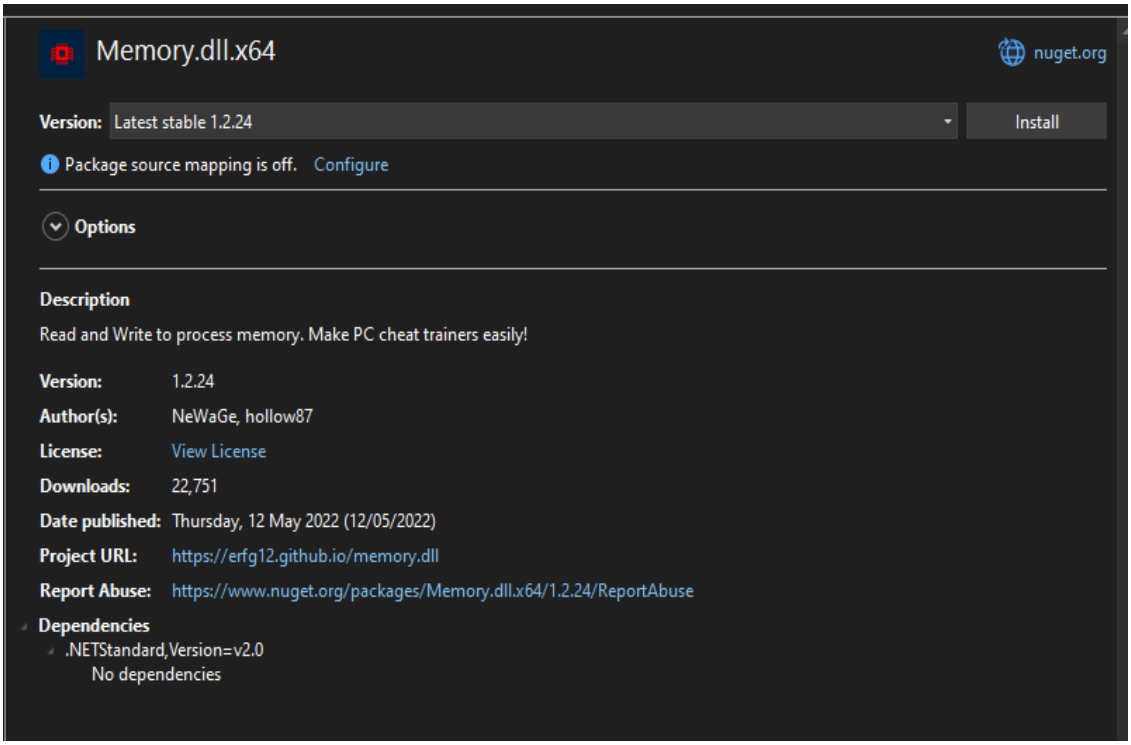
Slika 2: Visual Studio [12]

3.2.3 Windows Forms

Je ogrodje za izdelavo UI za C# .NET aplikacije. Omogoča potegni in spusti gradnjo uporabniškega vmesnika vmesnika in izgleda aplikacije. [13]

3.2.4 Memory.dll

Memory.dll je knjižnica, ki omogoča branje in pisanje v pomnilnik. Podpira vse glavne podatkovne tipe. V projekt jo lahko uvozimo preko NuGet upravitelja paketov, je orodje, ki omogoča preprosto iskanje, nameščanje in upravljanje zunanjih knjižnic (paketi), ki jih lahko uporabite v svojih C# projektih. [14]



Slika 3: Memory.dll (vir: lasten)

3.2.5 Swed.dll

Knjižnica Swed.dll (32 ali 64 bitna) je zelo podobna memory.dll, le da ima dodane funkcije za lažje branje nekaterih podatkovnih tipov iz pomnilnika. Primer je branje vektorjev. Z enim pomnilniškim naslovom (vsebuje eno decimalno število), lahko preberemo cel vektor naenkrat, ker so tri decimalne vrednosti vektorja v pomnilniku zapisane po vrsti (naslednja dva pomnilniška naslova od prvega). [15]

3.2.6 Unity

Unity je orodje za izdelavo 2D in 3D iger ter drugih aplikacij in programov, ki zahtevajo 3D grafike, lahko se uporablja tudi za izdelavo posnetkov in filmov. [16]

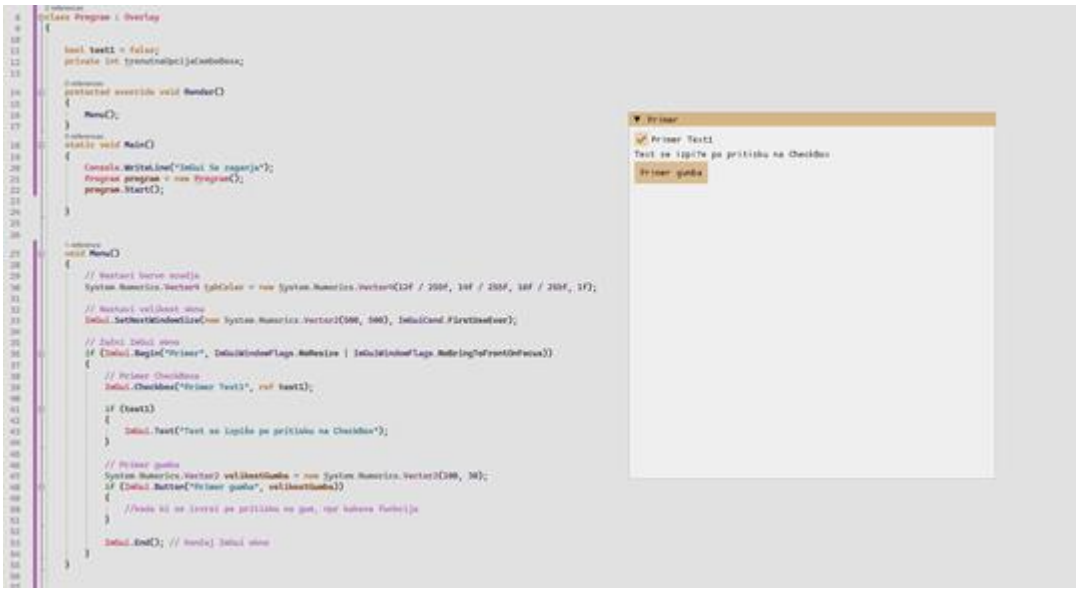
Slika 4: Unity logo [17]

Unity je "game engine", kar pomeni, da je to orodje primarno fokusirano na izdelavo iger. Za programiranje iger lahko uporabljamo C# ali povezujemo bloke oz. diagrame poteka ("visual scripting"). Prva verzija Unity je bila izdana leta 2005 z misijo, da naredi ustvarjanje iger bolj

dostopno. Skozi leta se je orodje razvijalo in prejemalo grafične, optimizacijske, zvokovne in simulacijske (simuliranje fizike) posodobitve. Danes je eno najbolj priljubljenih orodij za izdelavo iger in podpira vse večje platforme. [18]

3.2.7 Dear ImGUI

Dear ImGUI je programerska knjižnica, ki omogoča dodajanje UI in prilagojenega videza lastnemu programu. Sama koda je zelo podobna "C# Windows Forms", ima večjo prilagodljivost, vseeno pa manjka "drag and drop" funkcionalnost, ki bi omogočala programerjem, da dodajo elemente s predhodnim vizualnim pregledom, kako bo program izgledal. [19]



Slika 5: Uporaba Dear ImGUI (vir: lasten)

4 IZDELAVA IGRE S KOCKO

4.1 IGRA, V KATERI NE MOREMO GOLJUFATI

Assets > Kocka.cs > Kocka > VrziKocko

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using TMPro;
4   using Unity.VisualScripting;
5   using UnityEngine;
    3 references
6   public class Kocka : MonoBehaviour
7   {
        4 references
8       public int Vrednost {
9           get {…
46          set {
47              _vrednost = value;
48              // če je nova vrednost ista kot trenutna vrednost ne naredimo nič
49              if (value == Vrednost) return;
50              // zavrtimo kocko na določeno vrednost
51              if (value == 1)
52                  transform.rotation = Quaternion.Euler(0,-90.0f, 90.0f);
53              else if (value == 2)
54                  transform.rotation = Quaternion.Euler(0, 270f, 180f);
55              else if (value == 3)
56                  transform.rotation = Quaternion.Euler(-90f, 0, 90);
57              else if (value == 4)
58                  transform.rotation = Quaternion.Euler(90f, 0, -90f);
59              else if (value == 5)
60                  transform.rotation = Quaternion.Euler(0, 90f, 0);
61              else if (value == 6)
62                  transform.rotation = Quaternion.Euler(180f, 0, 90f);
63
64          }
65      }
```

Slika 6: Koda igre, v kateri ne moremo goljufati (vir: lasten)

Pri določenih igrah ali nekem delu igre ni mogoče zlorabljati delovnega pomnilnika, ker je koda napisana na način, ki tega ne dopušča oz. se ob tem, ko v delovnem pomnilniku neko vrednost spremenimo, nič ne zgodi. Primer tega je, da kodo za obračanje kocke damo v "set" metodo vrednosti.

"Set" metoda se sproži, kadar je vrednost spremenjena, a se ta sproži le s spreminjanjem iz kode, torej ko uporabljamo operator "=" in ne s spreminjanjamo vrednosti v pomnilniku iz nekega drugega programa. [20] "Set" metoda je uporabljena takrat, kadar želimo, da se ob spreminjanju vrednosti neke spremenljivke, izvede še neka dodatna funkcionalnost, ko npr. v zgornji sliki. Ta ob spreminjanju vrednosti spremenljivke "Vrednost" obrne kocko na to število.

"Set" metoda se v našem primeru ne sproži. Zaradi tega se v igri kocka ne obrne na želeno vrednost, čeprav je ta spremenjena v delovnem pomnilniku.

"Set" metoda se uporablja tudi v drugih jezikih, kot npr. v Javascriptu z ogrodjema React in Vue, kjer zagotavljajo tudi večjo varnost. [21]

Za goljufanje v takšnih igrah pa je pomemben in potreben drugačen pristop, najboljši primer je notranji program, ki nam v igro omogoča dodati svoj .dll in spremeniti kodo, ki pa spremeni

delovanje igre, a je lahko zelo enostavno zaznan. Če igralec z uporabo tega načina ni previden, je lahko kaznovan.

4.2 IGRA, V KATERI LAHKO GOLJUFAMO

Za izdelavo igre Met kocke sva izbrala program Unity, v katerega sva vstavila 3D objekt oz. "GameObject" za kocko. Tej kocki je nato bila dodana skripta, ki določi njeno vrednost ob pritisku tipke za presledek (angl. space key). Kocka se nato zavrti na naključno izbrano število. Koda za obračanje kocke se izvede vsak ukaz "Update()" oz. vsakič, ko se na ekran nariše nova sličica in se slika posodobi.

Vsak "Update()" torej primerjamo "Vrednost" in kocko obrnemo tako, da bo ploskev z ujemajočim se številom pik gledala navzgor. Kocko torej ves čas obračamo na vrednost

```
18      void Update()
19      {
20          // ob pritisku presledka izberemo naključno vrednost
21          if(Input.GetKeyDown(KeyCode.Space)) {
22              VrednostKocke = Random.Range(1, 6);
23          }
24
25          // posodobimo tekst na zaslonu
26          tekst.text = VrednostKocke.ToString();
27
28          //obrnemo kocko na določeno vrednost
29          if (VrednostKocke == 1)
30              transform.rotation = Quaternion.Euler(0,-90.0f, 90.0f);
31          else if (VrednostKocke == 2)
32              transform.rotation = Quaternion.Euler(0, 270f, 180f);
33          else if (VrednostKocke == 3)
34              transform.rotation = Quaternion.Euler(-90f, 0, 90);
35          else if (VrednostKocke == 4)
36              transform.rotation = Quaternion.Euler(90f, 0, -90f);
37          else if (VrednostKocke == 5)
38              transform.rotation = Quaternion.Euler(0, 90f, 0);
39          else if (VrednostKocke == 6)
40              transform.rotation = Quaternion.Euler(180f, 0, 90f);
41      }
42  }
43
```

Slika 7: Koda igre, v kateri lahko goljufamo (vir: lasten)

5 VPLIV KODE NA SPOSOBNOST GOLJUFANJA

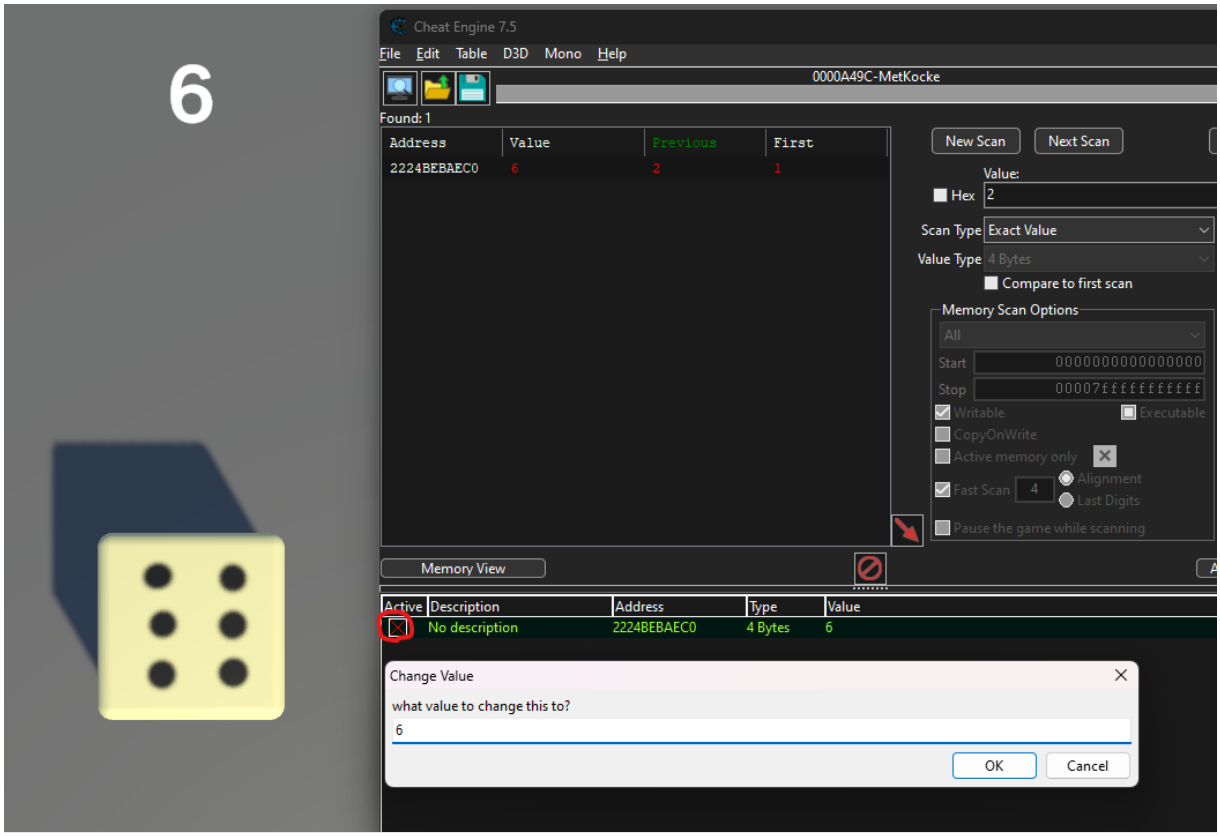
5.1 POSODABLJANJE NA VSAKO SLIČICO

Posodabljanje npr. rotacije kocke glede na spremenljivko "Vrednost" na vsako sličico je najbolj prepričljiv način goljufanja, še posebej če gre za "offline" igro. Kocka se obrne na pravo vrednost skoraj takoj, ko jo spremenimo v pomnilniku, saj se sličice posodabljajo več kot 60-krat na sekundo.

```
18      void Update()
19      {
20          // ob pritisku presledka izberemo naključno vrednost
21          if(Input.GetKeyDown(KeyCode.Space)) {
22              VrednostKocke = Random.Range(1, 6);
23          }
24
25          // posodobimo tekst na zaslonu
26          tekst.text = VrednostKocke.ToString();
27
28          //obrnemo kocko na določeno vrednost
29          if (VrednostKocke == 1)
30              transform.rotation = Quaternion.Euler(0,-90.0f, 90.0f);
31          else if (VrednostKocke == 2)
32              transform.rotation = Quaternion.Euler(0, 270f, 180f);
33          else if (VrednostKocke == 3)
34              transform.rotation = Quaternion.Euler(-90f, 0, 90);
35          else if (VrednostKocke == 4)
36              transform.rotation = Quaternion.Euler(90f, 0, -90f);
37          else if (VrednostKocke == 5)
38              transform.rotation = Quaternion.Euler(0, 90f, 0);
39          else if (VrednostKocke == 6)
40              transform.rotation = Quaternion.Euler(180f, 0, 90f);
41      }
42  }
43
```

Slika 8: Koda za kocko pri posodabljanju vsake sličice

Če pri taki igri uporabimo Cheat Engine in nastavimo vrednost kocke na recimo 6 in obkljukamo kvadratek ob vrednosti (kar naredi, da se cel čas nastavlja na 6), se nam bo kocka ves čas obračala na 6.



Slika 9: Spreminjanje vrednosti kocke v Cheat Enginu (vir: lasten)

5.2 POSODABLJANJE V "SET" METODI

Če npr. rotacijo kocke posodabljamo v "set" metodi spremenljivke "Vrednost", se bo ta obrnila le, kadar spremenljivki dodeljujemo novo število iz kode, z uporabo operatorja "=" (enačaj). Tako se kocka na ekranu ne bo obrnila na pravo vrednost, a ta vrednost bo še vseeno zapisana v pomnilniku in bo veljala, ko jo primerjamo npr. v "if" stavkih kode in uporabljamo v kalkulacijah (razen če se v tem primeru ves čas izvaja tudi metoda "get" – (slika 10).

Igro lahko naredimo tudi tako, da namesto tega, da kocko vrtimo mi, ta simulira fiziko z Unity komponento imenovano "Rigidbody". To nato ob pritisku gumba teleportiramo v zrak in ji dodamo silo in naključno začetno rotacijo. Kocka nato tako kot v resničnem svetu, pade na tla in se nekaj časa vrti.

Assets > Kocka.cs > Kocka > Vrednost

```
67      3 references
        public int _vrednost = 0;
68      2 references
        public Transform zacetekMeta;
69      1 reference
        public float mocMeta = 100;
70      1 reference
        public bool srediMeta = false;
71      1 reference
        public TextMeshProUGUI tekst;
72
73      0 references
        void Update()
74      {
75          var rb = this.GetComponent<Rigidbody>();
76
77          srediMeta = rb.angularVelocity.magnitude > 0.2f || rb.velocity.magnitude > 0.2f;
78
79          tekst.text = this.Vrednost.ToString();
80
81          //if(!srediMeta && Input.GetKeyDown(KeyCode.Space))
82          if(Input.GetKeyDown(KeyCode.Space))
83              VrziKocko();
84      }
        1 reference
85      public void VrziKocko() {
86          transform.position = zacetekMeta.position;
87          transform.rotation = Quaternion.Euler(Random.Range(0, 360), Random.Range(0, 360), Random.Range(0, 360));
88          this.GetComponent<Rigidbody>().velocity = zacetekMeta.forward * mocMeta * Random.Range(0.8f, 1.2f);
89      }
90  }
91
```

Slika 10: Koda za metanje kocke s simulacijo fizike (vir: lasten)

Assets > Kocka.cs > Kocka

```
using System.Collections;
using System.Collections.Generic;
using TMPro;
using Unity.VisualScripting;
using UnityEngine;
3 references
public class Kocka : MonoBehaviour
{
    4 references
    public int Vrednost {
        get {
            float min = -Mathf.Infinity;
            int vrednost = 0;

            if (Vector3.Dot(this.transform.right, Vector3.up) > min)
            {
                min = Vector3.Dot(this.transform.right, Vector3.up);
                vrednost = 1;
            }
            if (Vector3.Dot(-this.transform.up, Vector3.up) > min)
            {
                min = Vector3.Dot(-this.transform.up, Vector3.up);
                vrednost = 2;
            }
            if (Vector3.Dot(this.transform.forward, Vector3.up) > min)
            {
                min = Vector3.Dot(this.transform.forward, Vector3.up);
                vrednost = 3;
            }
            if (Vector3.Dot(-this.transform.forward, Vector3.up) > min)
            {
                min = Vector3.Dot(-this.transform.forward, Vector3.up);
                vrednost = 4;
            }
            if (Vector3.Dot(this.transform.up, Vector3.up) > min)
            {
                min = Vector3.Dot(this.transform.up, Vector3.up);
                vrednost = 5;
            }
            if (Vector3.Dot(-this.transform.right, Vector3.up) > min)
            {
                min = Vector3.Dot(-this.transform.right, Vector3.up);
                vrednost = 6;
            }
            _vrednost = vrednost;
            return _vrednost;
        }
        set {
            _vrednost = value;
            // če je nova vrednost ista kot trenutna vrednost ne naredimo nič
            if (value == Vrednost) return;
            // zavrtimo kocko na določeno vrednost
            if (value == 1)
                transform.rotation = Quaternion.Euler(0,-90.0f, 90.0f);
            else if (value == 2)
                transform.rotation = Quaternion.Euler(0, 270f, 180f);
            else if (value == 3)
                transform.rotation = Quaternion.Euler(-90f, 0, 90);
            else if (value == 4)
                transform.rotation = Quaternion.Euler(90f, 0, -90f);
            else if (value == 5)
                transform.rotation = Quaternion.Euler(0, 90f, 0);
            else if (value == 6)
                transform.rotation = Quaternion.Euler(180f, 0, 90f);

        }
    }
```

Slika 11: Koda za branje in nastavljanje rotacije kocke z metodama "set" in "get" (vir: lasten)

Pod spremenljivko "Vrednost" je tako bila dodana še metoda "get" (slika 11), ta se kliče vsako sličico, ker jo vsako sličico tudi "pridobivamo". Zato bo v pomnilniku še vedno zapisana prava vrednost in je ne bomo mogli spreminjati (oz. se bo takoj spremenila nazaj). V "get" torej pridobimo vrednost kocke s primerjanjem njenih "transform.up", " transform.right" in " transform.forward" smeri (vektorji za gor, desno in naprej) s smerjo (vektorjem) navzgor ("Vector3.up").

5.3 POSODABLJANJE V JAVNI METODI

Če npr. življenja igralca, ki so prikazana s srci, spremenimo na neko drugo vrednost, je možno, da se to ne bo videlo na ekranu. To se zgodi zato, ker se zaradi varčevanja in optimizacije ta posodobijo le, ko se v kodi pokličejo funkcije, ki nam življenja zmanjšajo ali povečajo. Primer tega je, da imamo v Igralec.cs (slika 12) skripti funkcijo "TakeDamage()". Ta od življenj odšteje vrednost, ki jo dobi, nato pa posodobi UI.

Torej če skozi naš program ali z uporabo Cheat Engina spremenimo "zivljenja" na recimo 99, se to ne bo prikazalo na ekranu, dokler se ne poškodujemo ali zdravimo.

Assets > Igralec.cs > ...

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using TMPro;
4   using Unity.VisualScripting;
5   using UnityEngine;
6
    0 references
7   public class Igralec : MonoBehaviour
8   {
        2 references
9       public int zivljenja = 3;
        1 reference
10      public TextMeshProUGUI tekst;
11
12
        0 references
13      void TakeDamage(int dmg) {
14          zivljenja -= dmg;
15          UpdateUI();
16      }
17
        1 reference
18      public void UpdateUI() {
19          tekst.text = "St preostalih zivljenj: " + zivljenja.ToString();
20      }
21  }
22
23
```

Slika 12: Primer optimizirane kode za življenja igralca (vir: lasten)

V primeru, da koda ni optimizirana, bi besedilo na zaslonu posodabljali tudi, ko se "zivljenja" niso spremenila. Če bi si "zivljenja" spremenili s programom ali Cheat Enginom, bi se to takoj videlo na ekranu. Koda za to bi izgledala tako:

Assets > Igralec.cs > ...

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using TMPro;
4   using Unity.VisualScripting;
5   using UnityEngine;
6
    0 references
7   public class Igralec : MonoBehaviour
8   {
        2 references
9       public int zivljenja = 3;
        1 reference
10      public TextMeshProUGUI tekst;
11
        0 references
12      void Update() => UpdateUI();
13
        0 references
14      void TakeDamage(int dmg) {
15          zivljenja -= dmg;
16          UpdateUI();
17      }
18
        2 references
19      public void UpdateUI() {
20          tekst.text = "St preostalih zivljenj: " + zivljenja.ToString();
21      }
22  }
23
```

Slika 13: Primer kode za življenja igralca, ki ves čas posodablja UI (vir: lasten)

5.4 POVEZAVA Z AVTORITATIVNIM STREŽNIKOM

Če bi naša igra s kocko bila za več igralcev prek interneta, bi najverjetneje uporabljala avtoritativni strežnik. Ob pritisku gumba za met, bi od strežnika prejeli novo "Vrednost". Če bi vrednost kocke spreminjali v pomnilniku, bi se ta spremenila le na našem ekranu in ne na strežniku in pri drugih igralcih. Tako je goljufanje pri večigralski igri prek interneta nemogoče z načinom spreminjanja vrednosti v pomnilniku.

6 ISKANJE NASLOVA V POMNILNIKU

6.1 KAJ JE POMNILNIŠKI NASLOV

Ko je spremenljivka ustvarjena, se spremenljivki dodeli pomnilniški naslov. Pomnilniški naslov je lokacija v delovnem pomnilniku, kjer je spremenljivka shranjena. Ko dodelimo neko vrednost spremenljivki, je ta vrednost tudi zapisana v tem pomnilniškem naslovu. [22]

6.2 KAJ JE KAZALNIK

Kazalnik je koncept v računalništvu, ki se uporablja v računalniški znanosti za sklicevanje ali kažipot na pomnilniško lokacijo, ki shranjuje vrednost ali objekt. Je spremenljivka, ki shranjuje naslov pomnilnika druge spremenljivke ali podatkovne strukture namesto samega podatka. [23]

6.3 ISKANJE NASLOVA V POMNILNIKU

Naslov iščemo tako, da preiščemo celoten pomnilniški prostor, ki ga zaseda aplikacija. Obdržimo pomnilniške naslove, ki vsebujejo vrednost, ki jo iščemo. Ko se ta vrednost spremeni, preiščemo rezultate prejšnjega iskanja in obdržimo tiste z željeno vrednostjo. Proces ponavljamo, dokler nam ne ostane samo en pomnilniški naslov.

Torej če imamo v igri trenutno 3 življenja, skeniramo vse naslove in obdržimo le tiste, v katerih je zapisana vrednost 3. Ko se vrednost spremeni npr. na 2, skeniramo preostale naslove in obdržimo vse, ki imajo vrednost 2. Te spremembe v večini primerov povzročamo z igranjem igre. Proces skeniranja ponavljamo, dokler nam ne ostane samo 1 naslov.

6.4 ISKANJE KAZALNIKOV NA NASLOVE

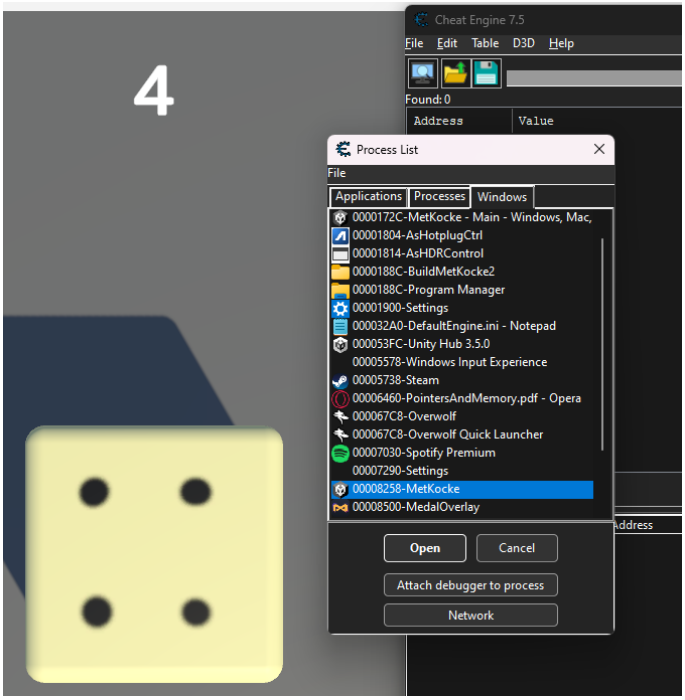
Vsak naslov ima zase tudi nekaj kazalcev – vrednosti, ki vsebujejo ta naslov. Ko igro ali aplikacijo ponovno zaženemo, se naslovi, ki smo jih prej našli, spremenijo. V njih torej ni več zapisano število življenj, ampak nekaj popolnoma drugega, naključnega.

Če želimo proces goljufanja pohitriti in izboljšati, teh naslovov ne želimo iskati vsakič, ko igro ali aplikacijo odpremo. Zato uporabimo kazalce ali "pointerje". Ti kažejo na naslov v pomnilniku. Za en naslov jih je lahko več, skoraj vedno pa je vsaj en stalen kazalnik, ki kaže na pravi naslov. [24]

6.5 ISKANJE NASLOVA ZA VREDNOST KOCKE

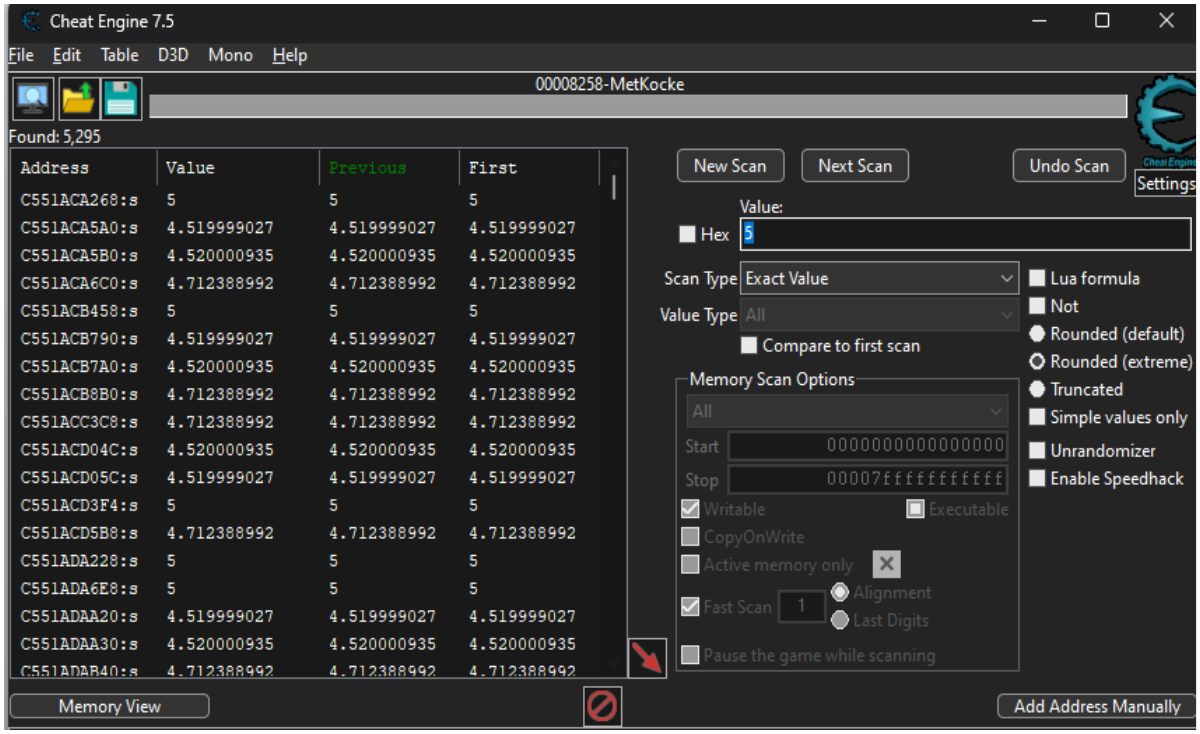
6.5.1 Iskanje naslova s Cheat Enginom

Ko zaženemo Cheat Engine, moramo izbrati proces, katerega pomnilniški prostor želimo skenirati za vrednosti.



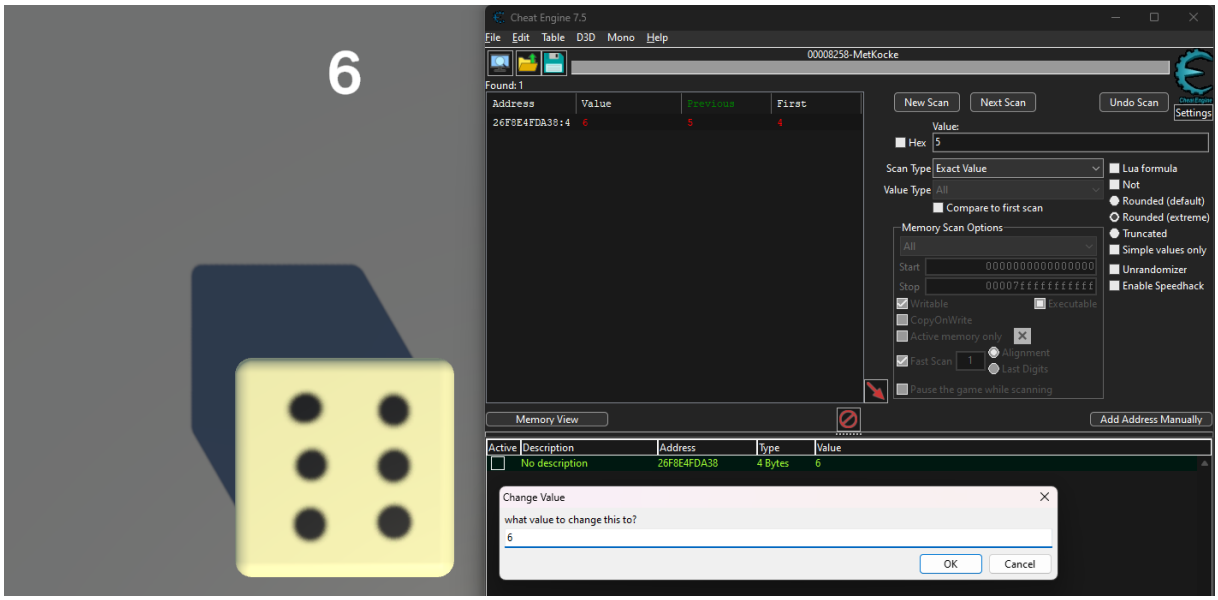
Slika 14: Izbiranje aplikacije za iskanje pomnilniških naslovov s programom Cheat Engine (vir: lasten)

Ko tega izberemo, lahko naredimo prvi sken za naslove, ki vsebujejo trenutno vrednost kocke, to je 4. Tako se nam na levem zavihku prikažejo vsi naslovi s to vrednostjo. Nato kocko vržemo še enkrat in skeniramo naslove za njeno novo vrednost. Proces ponavljamo, dokler nam ne ostane samo en naslov. [25]



Slika 15: Rezultati iskanja pomnilniških naslovov v Cheat Engin (vir: lasten)

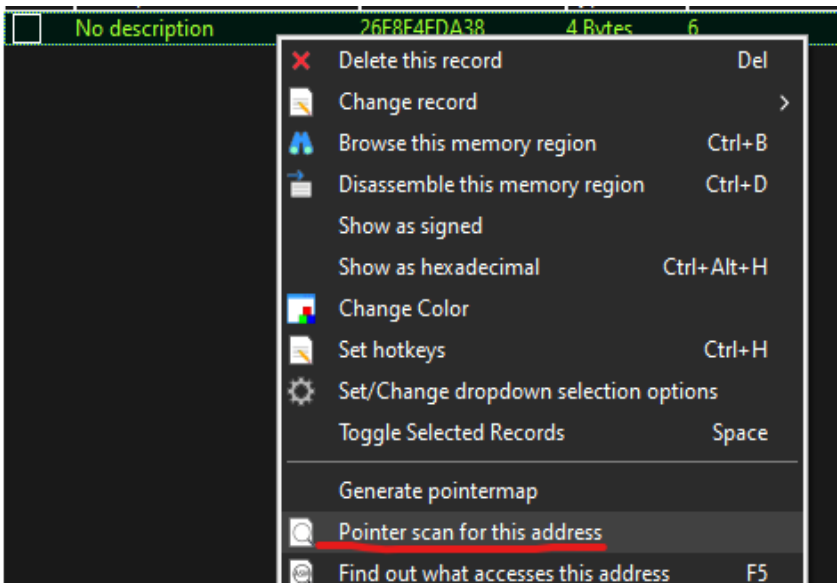
Ko najdemo naslov ki ga iščemo, lahko njegovo vrednost spremenimo kar v orodju Cheat Engine, s tem da nanj dvakrat kliknemo in ga dodamo v shranjene naslove. Nato ga še tam dvakrat kliknemo in vnesemo želeno vrednost. V našem primeru se kocka, ko vnesemo v naslov vrednost 6, res postavi na število 6.



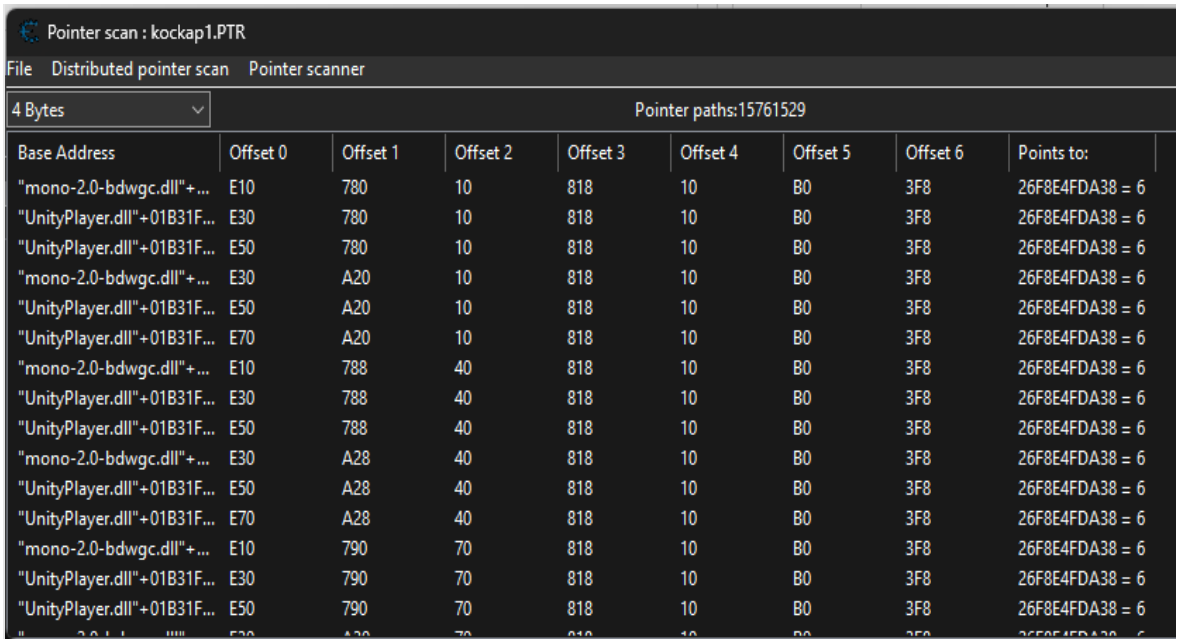
Slika 16: Nastavljanje vrednosti kocke v Cheat Engine (vir: lasten)

6.5.2 Iskanje kazalca na naslov

Ko imamo najden naslov, lahko najdemo tudi kazalnik, ki naslov za to vrednost vedno hrani. To naredimo s skeniranjem za kazalnike (angl. "pointer scan") ob desnem kliku na shranjen naslov. [24]



Slika 17: Skeniranje za kazalnike (vir: lasten)



Base Address	Offset 0	Offset 1	Offset 2	Offset 3	Offset 4	Offset 5	Offset 6	Points to:
"mono-2.0-bdwgc.dll"+...	E10	780	10	818	10	B0	3F8	26F8E4FDA38 = 6
"UnityPlayer.dll"+01B31F...	E30	780	10	818	10	B0	3F8	26F8E4FDA38 = 6
"UnityPlayer.dll"+01B31F...	E50	780	10	818	10	B0	3F8	26F8E4FDA38 = 6
"mono-2.0-bdwgc.dll"+...	E30	A20	10	818	10	B0	3F8	26F8E4FDA38 = 6
"UnityPlayer.dll"+01B31F...	E50	A20	10	818	10	B0	3F8	26F8E4FDA38 = 6
"UnityPlayer.dll"+01B31F...	E70	A20	10	818	10	B0	3F8	26F8E4FDA38 = 6
"mono-2.0-bdwgc.dll"+...	E10	788	40	818	10	B0	3F8	26F8E4FDA38 = 6
"UnityPlayer.dll"+01B31F...	E30	788	40	818	10	B0	3F8	26F8E4FDA38 = 6
"UnityPlayer.dll"+01B31F...	E50	788	40	818	10	B0	3F8	26F8E4FDA38 = 6
"mono-2.0-bdwgc.dll"+...	E30	A28	40	818	10	B0	3F8	26F8E4FDA38 = 6
"UnityPlayer.dll"+01B31F...	E50	A28	40	818	10	B0	3F8	26F8E4FDA38 = 6
"UnityPlayer.dll"+01B31F...	E70	A28	40	818	10	B0	3F8	26F8E4FDA38 = 6
"mono-2.0-bdwgc.dll"+...	E10	790	70	818	10	B0	3F8	26F8E4FDA38 = 6
"UnityPlayer.dll"+01B31F...	E30	790	70	818	10	B0	3F8	26F8E4FDA38 = 6
"UnityPlayer.dll"+01B31F...	E50	790	70	818	10	B0	3F8	26F8E4FDA38 = 6

Slika 18: Najdeni kazalniki

Pojavi se okno, z vsemi kazalci, ki kažejo na ta naslov. Ker je teh veliko, aplikacijo zapremo in ponovno odpremo ter skeniramo prej dobljene kazalce. Obdržimo vse, ki še vedno kažejo na trenutno vrednost kocke.

Dobimo kazalnik (lahko jih je več), ki bo vedno kazal na pravo vrednost (lahko se spremenijo, če računalniška igra prejme posodobitev).

7 IZDELAVA IN DELOVANJE ZUNANJEGA PROGRAMA

Vsaka igra ima dodeljene različne pomnilniške naslove, ki so shranjeni v bitih oz. binarni kodi (primer: 0111000111). Ti naslovi se med igranjem igre spreminjajo in vsebujejo različne lastnosti in števila (npr. število preostalih življenj, število virtualne valute, število preostalih življenjskih točk nasprotnikov itd.) V tem pomnilniškem prostoru so shranjeni vsi podatki, ki jih računalniška igra potrebuje za delovanje. Teh podatkov je zelo veliko, zato uporabjamo Cheat Engine, da lahko najdemo točno tistega, ki ga potrebujemo. Primer uporabe Cheat Engina je, da dobimo lokacijo, kjer je shranjeno število življenj.

Recimo, da ima naš karakter oz. igralec 100 življenskih točk, v Cheat Engine vpišemo, da iščemo vse pomnilnike, ki vsebujejo število 100 (skeniramo). Od tega dobimo veliko število kazalcev, ki kažejo na št.100. Naš naslednji korak je, da v igri to število spremenimo tako, da nas npr. nekaj udari, skočimo z višine in podobno. S tem smo dosegli, da se je naše število življenskih točk zmanjšalo, recimo da je padlo na 44.

Sedaj v Cheat Engine vnesemo število 44, program bo potem od tistih kazalcev ki so bili prej 100 izbral samo tiste, ki imajo sedaj vrednost 44. Ta korak lahko ponovimo, dokler nam ne ostane samo 1 kazalnik (večinoma 2x do 5x, odvisno od velikosti naše izbrane igre).

Sedaj smo dobili kazalnik, ki kaže, kje je to število zapisano v delovnem pomnilniku. Pametno je, da igro potem izklopimo in ponovno vklopimo, skeniramo za kazalnik, ki smo ga dobili pred izklopom ter ponovno skeniramo in s tem zagotovimo, da kazalnik ostane enak kljub ponovnem zagonu igre.

S tem procesom smo zdaj zagotovili in dobili pravilno shranjevalno mesto. In jo tudi shranimo, njen izgled pa je shranjen v pomnilniku in izgleda takole (Primer: 0x37F), ki pa je v šesnajstiškem (angl. hexadecimal) številu.

Te številke beremo v kodi.

```
Swed memory = new Swed("MetKocke");

IntPtr moduleBase = memory.GetModuleBase("UnityPlayer.dll");
IntPtr KockaRezultat = memory.ReadPointer(moduleBase, 0x01831FC8, 0xE98) + 0x38;
```

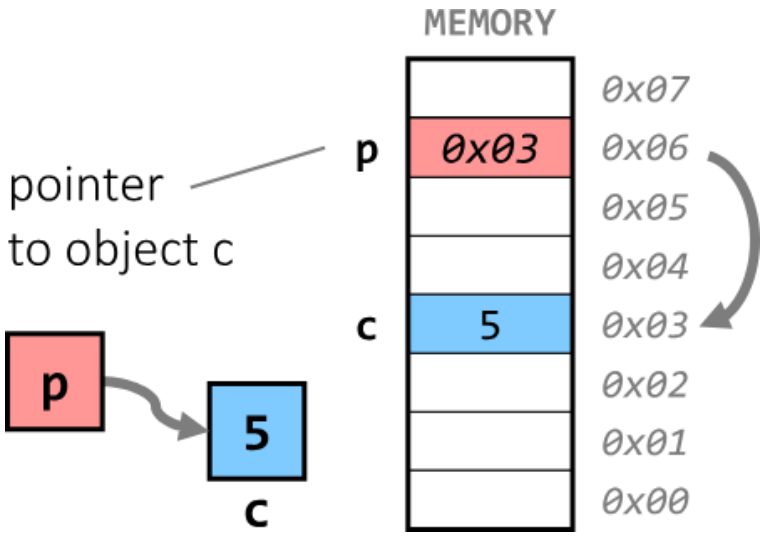
Slika 19: Dodaja kazalcev, ki kažejo in berejo pomnilnik (vir: lasten)

Na zgornji sliki vidimo 3 vrstice kode, ki so bile napisane v jeziku C#,

V prvi vrstici deklariramo, kateri prostor pomnilnika beremo ("MetKocke" je ime aplikacije oz. igre).

V drugi vrstici deklariramo kazalnik, ki shrani iz katere datoteke beremo pomnilnik (to dobimo z uporabo Cheat Engin).

V tretji vrstici pa deklariramo kazalnik, katerega beremo in vanj vnesemo naš pomnilniški naslov, v tem primeru ima 2 dodatna kazalca, ki ju dopišemo (0x1B31FC, 0xE98)+0x38.



Slika 20: Kazalniki v pomnilniku [26]

Branje ter zloraba tega pomnilnika je najlažji del pri izdelavi takšnega programa, saj do njega enostavno dostopamo z uporabo funkcije "WriteInt()", ki pa zahteva samo kazalnik, ki kaže na del pomnilnika, v katerem je ta informacija.

```
memory.WriteInt(KockaRezultat, izbera);
int tStanje = memory.ReadInt(KockaRezultat);
Thread.Sleep(100);
```

Slika 21: Branje stanja kocke (vir: lasten)

Branje lahko vidimo v vrstici št. 2, kjer ustvarimo novo "int" sprejemljivo z imenom "tStanje", ki pa uporablja funkcijo "ReadInt()" ter naslov prostora v pomnilniku, ki ga beremo in na koncu dodamo malo pavze.

Z dodatkom, ki omogoča uporabniku več kontrole nad programom, se ustvari program za goljufanje. V tem primeru nam omogoča, da izberemo število 6, na katerega se bo kocka obrnila.

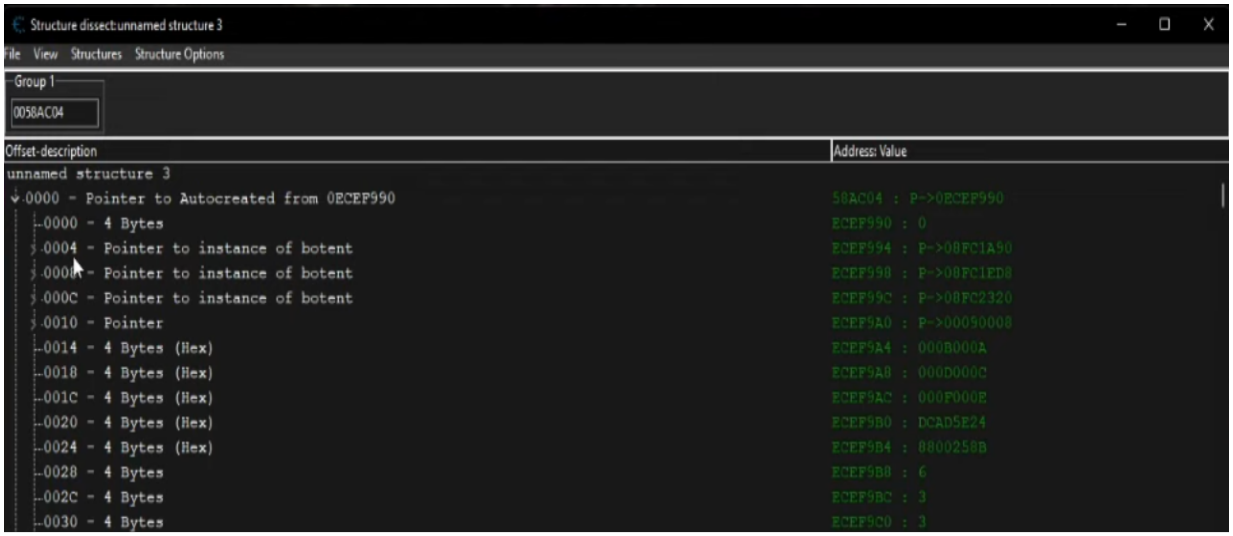
```
class Program
{
    static void Main()
    {
        Swed memory = new Swed("MetKocke"); //DEKLACACIJA PROGRAMA IZ KATEREGA BEREMO SPOMIN

        IntPtr moduleBase = memory.GetModuleBase("UnityPlayer.dll");

        IntPtr KockaRezultat = memory.ReadPointer(moduleBase, 0x01831FC8, 0xE98) + 0x38; //NAS KAZALEC ZA STEVILO KOCKE


        bool programRunning = true;

        while (programRunning)  //NEKONCAN LOOP KI ZA VEDNO KOCKA DA NA TO STEVILO
        {
            Console.WriteLine("VPISITE POLJUBEN REZULTAT KOCKE (Med 1 in 6) ali pritisnite 'x' za izhod:");


            string userInput = Console.ReadLine();  //PREBERE VNESENO STEVILO OD UPORABNIKA

            if (userInput.ToLower() == "x")
            {
                programRunning = false;
            }
            else
            {

                int izbera;
                if (int.TryParse(userInput, out izbera) && izbera >= 1 && izbera <= 6) //GLAVNI DEL PROGRAMA
                {
                    while (true)
                    {
                        memory.WriteInt(KockaRezultat, izbera); //ZAPIS V SPOMIN
                        int tStanje = memory.ReadInt(KockaRezultat);
                        Thread.Sleep(100);
                    }
                }
                else
                {
                    Console.Clear();
                    Console.WriteLine("Neveljavna izbira. Vnesite število med 1 in 6 ali pritisnite 'x' za izhod.");
                }
            }
        }
    }
}
```

Slika 22: Primer kode za vpliv na met kocke (vir: lasten)

8 BRANJE PODATKOV DRUGIH IGRALCEV

8.1 POMNILNIK V IGRAH

Igre so statično napisane (nekatere vrednosti imajo vedno iste naslove kazalnikov), s tem da uporabljajo prostor v pomnilniku, v katerem shranjujejo podatke vseh igralcev in predmetov. Do tega lahko dostopamo, enako kot bi dostopali do podatkov svojega igralca, saj se naši podatki tudi stranjujeno v enak prostor. Cheat Engine uporablja različne algoritme, da ugotovi če skupina kazalnikov kaže na podatke kot so "življenjske točke", "pozicija"... in označi, kaj predvideva, da ta skupina kazalnikov predstavlja. Na spodnji sliki lahko to vidimo na 3 skupnih vrsticah, na katerih piše, da kazalnik kaže na igralca.



Slika 23: Cheat Engine poglej v pomnilnik (vir: lasten)

8.2 PODATKI NA STRANI STREŽNIKA

V novejših igrah so nekateri podatki takšni, da do njih lahko dostopamo, a jih ne moremo spreminjati, saj pred vsako spremembo igre pošlje prošnjo strežniku, ki to preveri. V takšnem primeru lahko te podatke spreminjamo samo, kadar nismo povezani na internet kar pa nam v veliki večini igre ne dopušča igranja. V primerih, ko to lahko naredimo je, kadar igra zazna, da intertnet ni dostopen in te v igro da z umetno inteligenco oz. tako rečenimi "boti". [27]

8.3 USTVARJANJE ENTITY RAZREDA

```
0 references
public Entity ReadEntity(IntPtr entBase)
{
    var ent = new Entity();

    ent.baseAddress = entBase;


    ent.currentAmmo = mem.ReadInt(ent.baseAddress, Offsets.iCurrentAmmo);
    ent.health = mem.ReadInt(ent.baseAddress, Offsets.iHealth);
    ent.team = mem.ReadInt(ent.baseAddress,Offsets.iTeam);

    ent.feet = mem.ReadVector3(entBase, Offsets.vFeet);
    ent.head = mem.ReadVector3(entBase, Offsets.vHead);

    ent.name = Encoding.UTF8.GetString(mem.ReadBytes(ent.baseAddress, Offsets.sName, 11));

    return ent;
}
```

Slika 24: Ustvarjanje razreda nasprotnika (vir: lasten)

Ustvarila sva novi "Entity", ki pa bo obdržal vse pomembne podatke za vsakega igralca, kot so npr. število njihovih preostalih metkov, njihove življenske točke, v kateri skupini so ter njihovo pozicijo. Pomebno je dodati, da je branje njihovega imaga bolj napredno, saj, ne vemo ali imajo igralci kakšne posebne črke, ki jih ASCII program ne zazna. To je zelo pogosto in v tem primeru

ni optimalno. Dodamo lahko tudi "if" stavek, ki nenavadne simbole spremeni v "$" ali karkoli drugega.

8.4 BRANJE IN DODAJANJE IGRALCEV V ENTITY

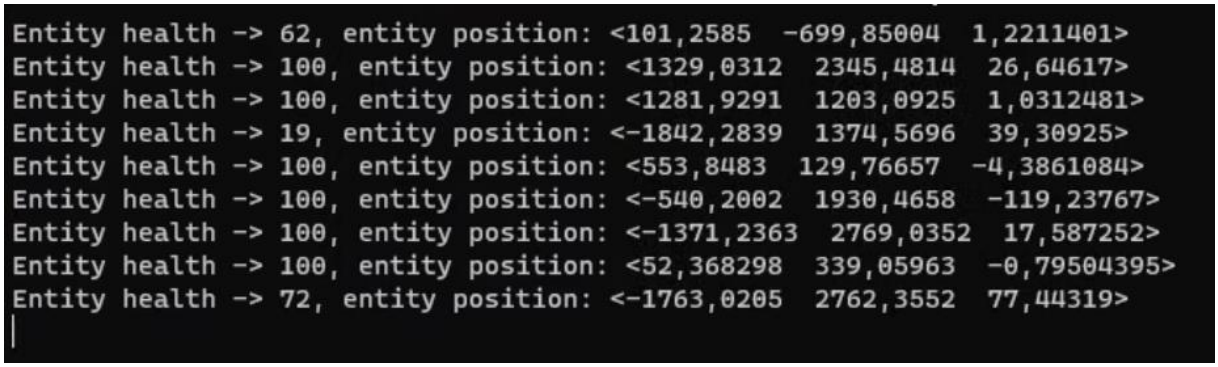
```
0 references
public List<Entity> ReadEntities(Entity localPlayer)
{
    var entities = new List<Entity>();
    var entityList = mem.ReadPointer(moduleBase,Offsets.iEntityList);

    for (int i = 0; i  < 4; i++)
    {
        var currentEntBase = mem.ReadPointer(entityList, i * 0x4);
        var ent = ReadEntity(currentEntBase);
        entities.Add(ent);   Tab to accept
    }
}
```

Slika 25: Ustvarjanje lista nasprotnikov (vir: lasten)

Entity deklariramo in shranimo v prostor v jeziku C# s pomočjo seznama (angl. list). Dodamo lahko tudi različne pogoje z "if" stavki za preverjanje življenja igralca pred dodajanjem v seznam. Na koncu uporabimo zanko"for", ki bo dodajala igralce v seznam, dokler ne dosežemo izbranega števila, na primer 4.

Pod "for" zanko pa lahko vidimo "currentEntBase", ki je tipa "var" (v resnici ni podatkovni tip, a se določi sam). Ime je poljubno in ga lahko zamenjamo. Kot že vemo, so vsi igralci shranjeni pod istim kazalcem, a niso na enakem mestu. V tem delu se igre tudi razlikujejo. V tem primeru je igraec od drugega shranjen 0x4 mest stran. To lahko vidimo tudi na začetku tega poglavja na levi strani. [28]

Uporaba pridobljenih podatkovTa sistem je zelo priljubjen, enostaven in lep na pogled. Omogoča nam tudi zelo lahko uporabo teh podatkov. Kor primer bi lahko v main funkciji dodali, da izpisuje ent.helath, kar bi nam izpisalo 9 vrstic, ki bi vsebovale 9 različnih podatkov, ki prikazujejo število življenskih točk in njihovo pozicijo..



Slika 26: Izpis nasprotnikov v konzoli (vir: lasten)

8.5 KAJ VSE LAHKO NAREDIMO S PRIDOBLJENIMI PODATKI

S temi podatki lahko izdelamo aplikacijo oz. orodje za goljufanje v igri, za UI lahko uporabimo Dear ImGUI.

Uporaba teh podatkov lahko da igralcu veliko prednost pred ostalimi, že podatki, ki jih lahko vidimo v konzoli (zgornja slika), bi bili zelo uporabni, a to lahko nadgradimo z različnimi metodami.

8.6 "AIMBOT"

"Aimbot" je program, ki ob streljanju avtomatsko nameri na nasprotnika. To lahko enostavno naredimo tako, da beremo pozicijo najbližjega nasprotnika.

Nadaljujemo tako, da beremo tudi rotacijo pogleda oz. glave igralca, ki jo ob pritisku gumba za streljanje spremenimo tako, da je obrnjena proti najbližjemu nasprotniku. [29]

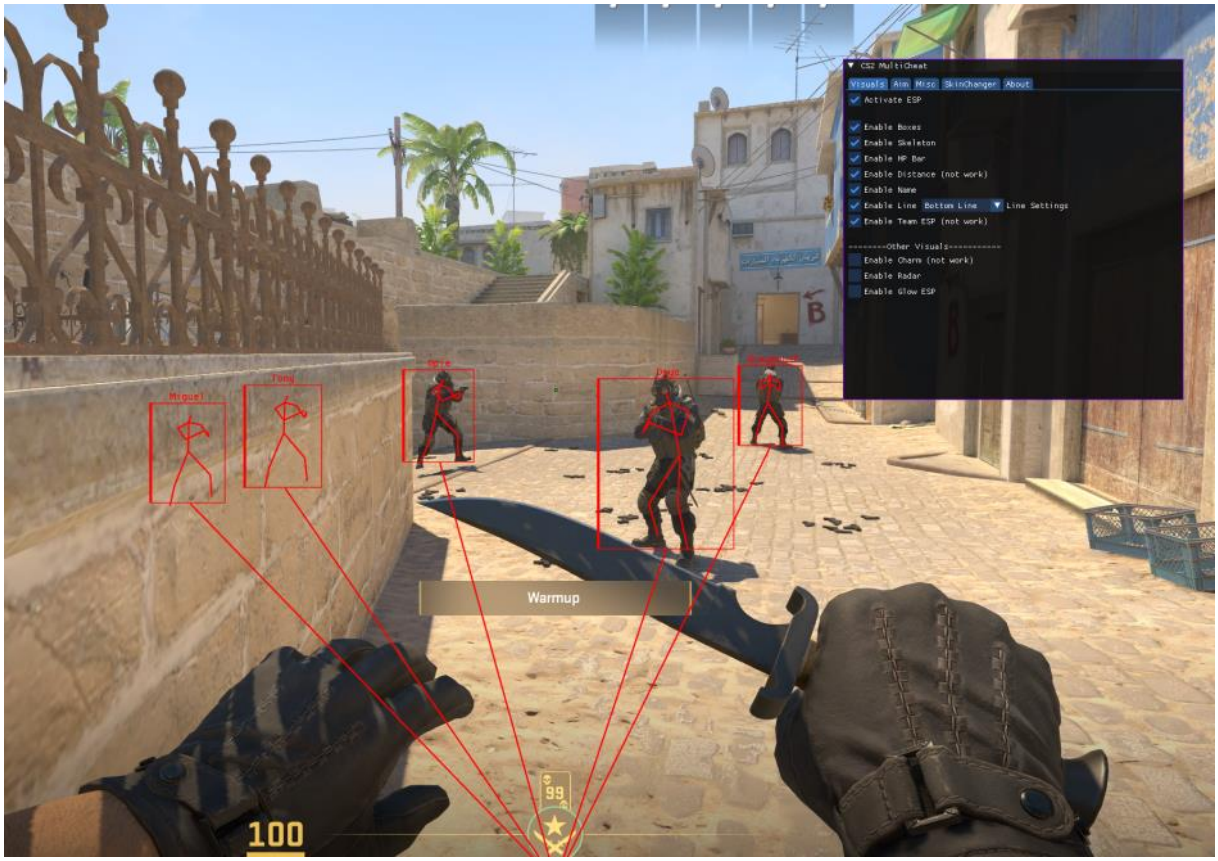
8.7 "TRIGGERBOT"

Je program, ki zazna, ko je nasprotnik na merilni piki (angl. crosshair) igralca in v tem primeru takoj ustreli nasportnika. Ima zelo enostavno kodo, saj ima vsak nasprotnik določen ID, ki ga lahko beremo. V konstantni zanki preverjamo, če je igralčeva merilna pika na nasprotniku. V primeru, da je, simuliramo klik miške.

Uporabniku damo možnost dodajanja časovnega zamika med zaznavanjem in strelom, saj imamo ljudje reakcijski čas in bi v nasprotnem primeru bilo preveč očitno, da goljufamo. To lahko naredimo z enostavno "Thread.Sleep()" funkcijo, ki zaustavi kodo za določen čas. Ta funcija potrebuje osnovno knjižnico (System.Threading).

8.8 ESP

Je vsaka vrsta programa, ki igralcu daje kakršno koli obliko informacij, glede tega, kje se nahajajo nasprotniki. Je najbolj popularna verzija goljufanja, a zelo težka za izvedbo. Njena najmanj zahtevna oblika je samo izpis koordinat nasprotnikov v konzolo.



Slika 27: Primer osebnega projekta (vir: lasten)

Program lahko nadgradimo tako, da uporabimo plast z brezbarvnim ozadjem (angl. overlay), ki je narisana čez okno programa. Na to plast nato narišemo okvirje. V ozadju je veliko matematike, saj imamo v igri 3 koordinate, ko rišemo na ekran pa imamo samo 2. Za to uporabimo funkcijo "world to screen", ki uporabi velikost zaslona igralca, njegovo trenutno pozicijo, pozicijo nasprotnikov in resolucijo igre oz. okna programa, da izračuna koordinate slikovnih pik predmeta iz igre, v našem primeru nasprotnikov.

Te 3 metode so najpogosteje uporabljene, seveda lahko s takšnimi informacijami naredimo veliko več stvari, ki pa ni nujno, da so tako koristne za goljufanje.

```
3 references
Vector2 WorldToScreen(ViewMatrix matrix, Vector3 pos, int width, int height)
{
    Vector2 screenCoordinates = new Vector2();

    float screenW = (matrix.m41 * pos.X) + (matrix.m42 * pos.Y) + (matrix.m43 * pos.Z) + matrix.m44;

    if (screenW > 0.001f)
    {
        float screenX = (matrix.m11 * pos.X) + (matrix.m12 * pos.Y) + (matrix.m13 * pos.Z) + matrix.m14;
        float screenY = (matrix.m21 * pos.X) + (matrix.m22 * pos.Y) + (matrix.m23 * pos.Z) + matrix.m24;

        float camX = width / 2;
        float camY = height / 2;

        float X = camX + (camX * screenX / screenW);
        float Y = camY - (camY * screenY / screenW);

        screenCoordinates.X = X;
        screenCoordinates.Y = Y;
        return screenCoordinates;
    }
    else
    {
        return new Vector2(-99, -99);
    }
}
```

Slika 28: View matrix (vir: lasten)

Pomembno je omeniti, da program uporablja tudi informacije od "ViewMatrix", ki pa je v računalniški grafiki matrika, ki predstavlja transformacijo pogleda ali kamere v tridimenzionalnem prostoru, običajno je uporabljena za opis ogleda scene v računalniških igrah ali simulacijah.

9 RAZPRAVA

1. **Z branjem in spreminjanjem delovnega pomnilnika lahko vplivamo na igro Met kocke.**

V igri met kocke sva lahko z uporabo zunanjega programa za goljufanje, s pisanjem v pomnilnik, uspešno vplivala na izid meta kocke.

Da povzameva, ta hipoteza je **potrjena**, saj s spreminjanjem pomnilnika igre lahko vplivamo na igro Met kocke.

2. **Tehnologija, ki je uporabljena pri razvoju igre, vpliva na našo zmožnost zlorabe delovnega pomnilnika?**

Hipotezo lahko **potrdimo**, saj tehnologija oz. koda res vpliva na našo zmožnost goljufanja. To, kako je program napisan, nam lahko goljufanje tudi oteži.

3. **V igri lahko beremo podatke nasprotnikov**

Zadnja hipoteza zahteva veliko eksperimentiranja v veliko različnih igrah, za raziskavo sva uporabila igro “Counter Strike 2”, ki je znana po svoji slabi zaščiti.

Ugotovila sva, da je brati podatke drugih igralcev presenetljivo podobno branju lastnih podatkov in da izvedba ni tako zahtevna, kot se je sprva zdelo. Zato lahko hipotezo **potrdimo**.

10 ZAKLJUČEK

V raziskovalno nalogo sva se podala s solidnim predznanjem programiranja ter izrazitim zanimanjem za področje računalniških iger. Pritegnilo naju je vprašanje, kako lahko izboljšamo svojo izkušnjo pri igranju iger z dodajanjem funkcij ali celo goljufanjem v igrah. Skozi izvajanje raziskave sva pridobila dragocene vpoglede v kompleksnost računalniških iger in mehanizmov, ki omogočajo goljufanje. Ugotovila sva, da je razvoj tehnologije prinesel nove načine, kako lahko igralci manipulirajo s podatki v igrah, hkrati pa sva postala bolj ozaveščena o etičnem ravnanju v igralni skupnosti.

Nadaljnje delo na tem področju bi lahko vključevalo nadaljnje raziskave in eksperimentiranje z različnimi metodami za preprečevanje goljufanja ter izboljšanje varnosti in poštenosti v spletnih in večigralskih igrah. Pomembno je tudi poudariti ozaveščanje igralcev o posledicah goljufanja ter spodbujanje k spoštovanju pravil in etičnih načel v igralni skupnosti. S tem bi lahko prispevali k boljšemu in bolj poštenemu igralnemu okolju za vse udeležene.

11 POVZETEK

Igranje računalniških iger je vedno bolj popularna aktivnost, še posebej med mladimi. Okoli tega je nastala cela skupnost entuziastičnih igralcev, ki v tako imenovanih "e-športih" tekmujejo tudi za denar in druge nagrade. Tako kot pri drugih računalniških področjih, je elektronske naprave možno tudi zlorabljati, to pa nekateri igralci računalniških iger naredijo, da lahko pri igranju goljufajo. V tej raziskovalni nalogi sva se poglobila v goljufanje v računalniških igrah z zlorabo in spreminjanjem vrednosti v pomnilniku.

Zanimalo naju je branje pomnilnika, iskanje pomnilniških naslovov podatkov igre in vpliv kode na zmožnost goljufanja, koriščenje pridobljenih podatkov in vplivanje.

Izdelala in testirala sva igro, v kateri mečeš kocko in s spreminjanjem vrednosti v pomnilniku vplivala na rezultat meta. Potrdila sva, da tehnologija, ki jo uporablja program oz. način, na katerega je napisan program, res vpliva na našo zmožnost goljufanja pri uporabi zunanjih programov za goljufanje in da lahko beremo podatke nasprotnikov, kot so pozicija in življenjske točke. Te podatke je možno uporabiti tudi na več načinov, kot so avtomatsko merjenje na nasprotnike, avtomatsko streljanje nasprotnikov in to, da vidimo nasprotnike skozi stene.

12 ZAHVALA

Zahvaljujeva se mentorju Islamu Mušiću za pomoč in usmerjanje pri raziskovalni nalogi in pa dr. Nataši Meh Peer za lektoriranje naloge. Zahvaljujeva se tudi sošolcu Adrianu Karahmetoviću za povratne informacije ob branju naloge ter staršem za vso njihovo podporo ob najinem raziskovanju.

13 VIRI IN LITERATURA

[1] „The entire history of game cheating,“ [Elektronski]. Available: https://www.popularmechanics.com/culture/gaming/a33650224/cheat-code-history/. [Poskus dostopa 24 12 2023].

[2] „What is an arduino,“ [Elektronski]. Available: https://learn.sparkfun.com/tutorials/what-is-an-arduino/all. [Poskus dostopa 20 2 2024].

[3] „Arduino mouse click,“ [Elektronski]. Available: https://www.arduino.cc/reference/cs/language/functions/usb/mouse/mouseclick/. [Poskus dostopa 20 2 2024].

[4] „What is raspberry pi,“ [Elektronski]. Available: https://www.spiceworks.com/tech/networking/articles/what-is-raspberry-pi/ . [Poskus dostopa 10 2 2024].

[5] „Valve Anti-Cheat,“ [Elektronski]. Available: https://developer.valvesoftware.com/wiki/Valve_Anti-Cheat. [Poskus dostopa 13 2 2024].

[6] „What is Vanguard?,“ [Elektronski]. Available: https://support-valorant.riotgames.com/hc/en-us/articles/360046160933-What-is-Vanguard. [Poskus dostopa 20 2 2024].

[7] „Easy Anti Cheat,“ [Elektronski]. Available: https://www.easy.ac/en-us/. [Poskus dostopa 13 2 2024].

[8] G. Gabriel, „Fast-Paced Multiplayer (Part I): Client-Server Game Architecture,“ [Elektronski]. Available: https://www.gabrielgambetta.com/client-server-game-architecture.html. [Poskus dostopa 25 1 2024].

[9] D. Kenneth, „The Role of Authorotative Dedicated Servers in Live Game Development,“ [Elektronski]. Available: https://accelbyte.io/blog/the-role-of-authoritative-dedicated-servers-in-live-game-development. [Poskus dostopa 25 1 2024].

[10] „Cheat Engine, About,“ [Elektronski]. Available: https://www.cheatengine.org . [Poskus dostopa 9 12 2023].

[11] „Visual Studio, About,“ [Elektronski]. Available: https://visualstudio.microsoft.com/ . [Poskus dostopa 11 12 2023].

[12] „Visual Studio (slika),“ [Elektronski]. Available: https://technewsspace.com/wp-content/uploads/2023/08/Microsoft-will-close-Visual-Studio-for-Mac-service-updates.jpg. [Poskus dostopa 16 2 2024].

[13] „Desktop Guide (Windows Forms .NET),“ [Elektronski]. Available: https://learn.microsoft.com/en-us/dotnet/desktop/winforms/overview/?view=netdesktop-8.0 . [Poskus dostopa 12 11 2023].

[14] „Memory.dll Github stran,“ [Elektronski]. Available: https://github.com/erfg12/memory.dll/. [Poskus dostopa 13 12 2023].

[15] „Swed.dll Github stran,“ [Elektronski]. Available: https://github.com/Massivetwat/swed32. [Poskus dostopa 10 12 2023].

[16] „Unity.com,“ [Elektronski]. Available: https://unity.com/. [Poskus dostopa 1 2 2024].

[17] „Unity logotip,“ [Elektronski]. Available: https://www.google.com/url?sa=i&url=https%3A%2F%2Fen.wikipedia.org%2Fwiki%2FUnity_%2528game_engine%2529&psig=AOvVaw0XHPldtMqfQN7LWaLjNgTZ&ust=1708109497446000&source=images&cd=vfe&opi=89978449&ved=2ahUKEwjEnffvga6EAxUcgP0HHUf7C-8QjRx6BAgAEBc. [Poskus dostopa 13 2 2024].

[18] „Unity (game engine),“ [Elektronski]. Available: https://en.wikipedia.org/wiki/Unity_(game_engine). [Poskus dostopa 14 2 2024].

[19] „Dear ImGUI Github stran,“ [Elektronski]. Available: https://github.com/ocornut/imgui. [Poskus dostopa 4 2 2024].

[20] „C# properties - get in set,“ [Elektronski]. Available: https://www.w3schools.com/cs/cs_properties.php. [Poskus dostopa 1 2 2024].

[21] „JavaScript Getter and Setter,“ [Elektronski]. Available: https://www.scaler.com/topics/javascript/getter-and-setter-in-javascript/. [Poskus dostopa 15 2 2024].

[22] „w3schools.com,“ [Elektronski]. [Poskus dostopa 15 2 2024].

[23] „Kazalniki,“ [Elektronski]. Available: https://en.wikipedia.org/wiki/Pointer_(computer_programming). [Poskus dostopa 10 2 2024].

[24] „Cheat Engine Help - Pointer Scan,“ [Elektronski]. Available: https://cheatengine.org/help/pointer-scan.htm. [Poskus dostopa 11 2 2024].

[25] „Cheat Engine Forum - Tutorial,“ 11 2 2024. [Elektronski].

[26] „hackingcpp.com,“ [Elektronski]. Available: https://hackingcpp.com/cpp/lang/pointers1.svg. [Poskus dostopa 1 2 2024].

[27] „Difference between Client & Server Sided,“ [Elektronski]. Available: https://forum.cheatengine.org/viewtopic.php?t=101761 . [Poskus dostopa 15 2 2024].

[28] „How Read The Entity List For Any Game, But Also Manage it | C# [Tutorial],“ [Elektronski]. Available: https://www.youtube.com/watch?v=_qAwzlC38CE. [Poskus dostopa 15 2 2024].

[29] „How aimbots work (YouTube),“ [Elektronski]. Available: https://www.youtube.com/watch?v=FM1E0ucndFw. [Poskus dostopa 20 2 2024].