# Razvoj trodimenzionalne računalne igre u programskom alatu Unity

**Hamzić, Domagoj**

**SVEUČILIŠTE U ZAGREBU**

**FAKULTET ORGANIZACIJE I INFORMATIKE**

**V A R A Ž D I N**

**Domagoj Hamzić**

# RAZVOJ TRODIMENZIONALNE RAČUNALNE IGRE U PROGRAMSKOM ALATU UNITY

**DIPLOMSKI RAD**

**Varaždin, 2021.**

**SVEUČILIŠTE U ZAGREBU**

**FAKULTET ORGANIZACIJE I INFORMATIKE**

**V A R A Ž D I N**

**Domagoj Hamzić**

**Matični broj: 44242/15–R**

**Studij: Informacijsko i programsko inženjerstvo**

# RAZVOJ TRODIMENZIONALNE RAČUNALNE IGRE U PROGRAMSKOM ALATU UNITY

**DIPLOMSKI RAD**

**Mentor:**

Doc. dr. sc. Konecki Mario

**Varaždin, rujan 2021.**

*Domagoj Hamzić*

**Izjava o izvornosti**

Izjavljujem da je moj diplomski rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

*Autor/Autorica potvrdio/potvrdila prihvaćanjem odredbi u sustavu FOI-radovi*

**Sažetak**

Ovaj diplomski rad baziran je na osnovama kreiranja trodimenzionalne računalne igre te kreiranju grafike tj. objekata u alatu za trodimenzionalno modeliranje. Može se reći da je 90% rada praktične prirode, a ostatak je dokumentiranje kreiranog. Rad je rađen pomoću dva alata, Unity i Blender. Unity je služio za kreiranje računalne igre, logike dok je Blender korišten za kreiranje trodimenzionalnih objekata koji su uneseni u samu igru. Svi objekti koji imaju fizičko ponašanje i animaciju su vlastoručno kreirani dok su statični objekti preuzeti te su navedeni izvori preuzimanja istih. Prije praktičnog djela rada tj. izrade modela i same igre, ukratko ću proći kroz osnove Blendera i Unitya. Nakon toga slijedi objašnjenje kreiranja pojedinih trodimenzionalnih modela, prolazak kroz osnovne programske skripte koje predstavljaju logiku igre te opis igranja igre i rješavanja zagonetki.

# Sadržaj

# 1. Uvod

Tema diplomskog rada je, kao što kaže i sam naslov, kreiranje trodimenzionalne igre u programskom alatu, Unity. Tema je izabrana zbog simpatija prema igranju računalnih igara od samog djetinjstva te znatiželji što se sve nalazi „ispod haube“ jedne cjelovite igre. Isto tako, za završni rad bavio sam se trodimenzionalnim modeliranjem objekata te sam htio spojiti to znanje sa kreiranjem same igre tako da je lako zaključiti da su svi korišteni objekti u igri kreirani u programskom alatu za trodimenzionalno modeliranje, Blender. Motivacija također leži u izazovu i učenju novih stvari jer su ovakvi radovi idealni za učenje novih stvari i proširenju palete IT znanja.

# 2. Opis korištenih alata

U samom sažetku diplomskog rada navedeni su alati koji su korišteni prilikom kreiranja diplomskog rada. To su sljedeći alati:

- Unity
- Blender

Slijedi kratki opis navedenih alata.

## 2.1. Unity

Unity je besplatni alat za razvijanje igrica koji je idealan za početnike jer je prilično jednostavan za korištenje te ima ogroman broj online tečajeva pomoću kojih se mogu naučiti početničke, ali i naprednije tehnike korištenja Unitya. Za razvoj ove igre, koristio sam dva online tečaja koji su kupljeni na web aplikaciji čija je glavna svrha pružanje online tečajeva, Udemy. Slijede poveznice na dva tečaja koji su korišteni:

- Build A Multiplayer Kart Racing Game in Unity
- Master Procedural Maze & Dungeon Generation

Unity podržava 3D i 2D razvijanje računalnih igara. Isto tako nudi moćan IDE jer omogućuje mnoge značajke koje su povezane sa fizikom, prikazom igre i sl. Što se tiče programskog jezika i kreiranje same logike igre, Unity koristi C# programski jezik. Alat nudi mnoge klase i API-e koje je potrebno naučiti kako bi se što uspješnije i jednostavnije radila logika igre.
Za potrebe kreiranja igre, korištena je Unity 2020.1.0f1 verzija alata koju je moguće preuzeti na sljedećoj poveznici: https://unity3d.com/get-unity/download/archive
Verzija je izabrana zbog lakšeg praćena ranije navedenih tutorijala.

## 2.2. Blender

Blender pripada skupini alata za trodimenzionalno modeliranje. Korišten je zato što je među jednostavnijim alatima za modeliranje te je zbog toga idealan za početnike, a i odlično je podržan od strane Unitya. Riječ je o besplatnom open source alatu koji se koristi za izradu grafike računalnih igara, animiranih filmova, vizualnih efekata i sl. Alat sadrži mnoge mogućnosti za korištenje, a za izradu ovog rada korišteni su sljedeći:

- Modeliranje
- Teksturiranje
- Animiranje

# 3. Escape room

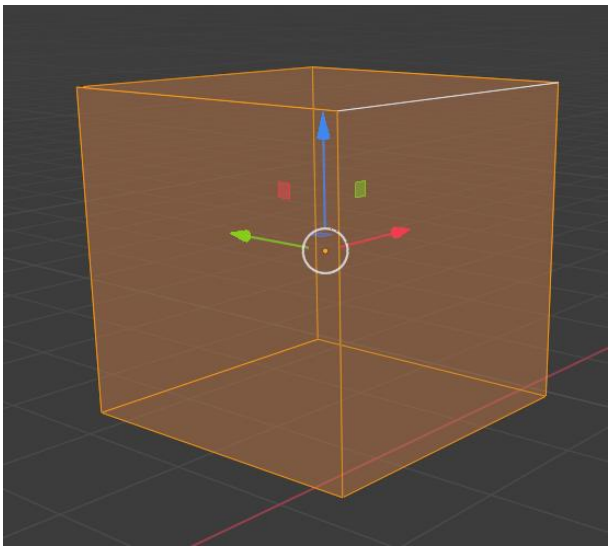
Sam naziv poglavlja otkriva o kakvoj je igri riječ. Cilj igre je izaći iz sobe u kojoj se nalaze zagonetke koje je potrebno riješiti. Glavni junak igre je implementiran kao „First-person“ tj. grafička perspektiva prikazana s gledišta karaktera igrača. Struktura poglavlja raspodijeljena je u dva dijela, a to su kreiranje modela, animacija u alatu Blender te unos istih i kreiranje poslovne logike igre.

## 3.1. Kreiranje modela

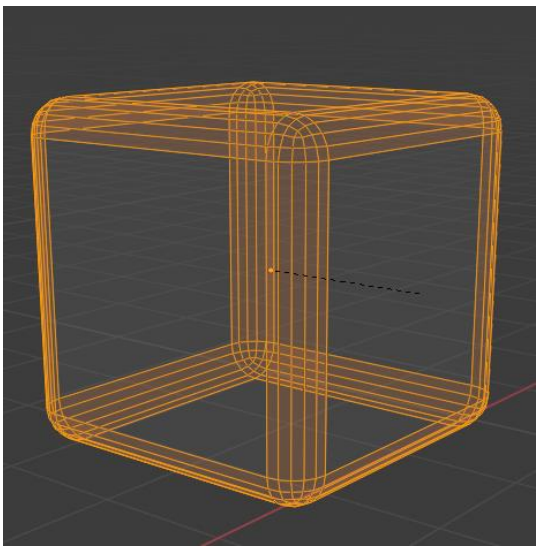
Za početak je potrebno kreirati trodimenzionalne modele koji će se koristiti u igri. Prvo se kreira oblik modela, nakon čega slijedi teksturiranje i animiranje modela. Slijedi opis kreiranja modela, teksturiranja i animiranja modela.

### 3.1.1. Oblik modela

Ladicu čine dva osnovna objekta, kocka za osnovni oblik ladice te kugla za noge ladice. Prvo se doda osnovni oblik kocke (Add->Mesh->Cube). Cilj je da tijelo ladice ima zaobljene rubove. To se radi na sljedeći način, prvo je potrebno prebaciti upravljanje objekta sa Object Mode u Edit Mode te postaviti odabir tijela objekta u Edge Select. Pritiskom tipke „A“ označavaju se svi rubovi objekta. Da bi se rubovi izgladili potrebno je pritisnuti sljedeću kombinaciju na tipkovnici „Ctrl+B“ (Edge->Bevel Edges) te sa mišem podesiti željenu zaobljenost.

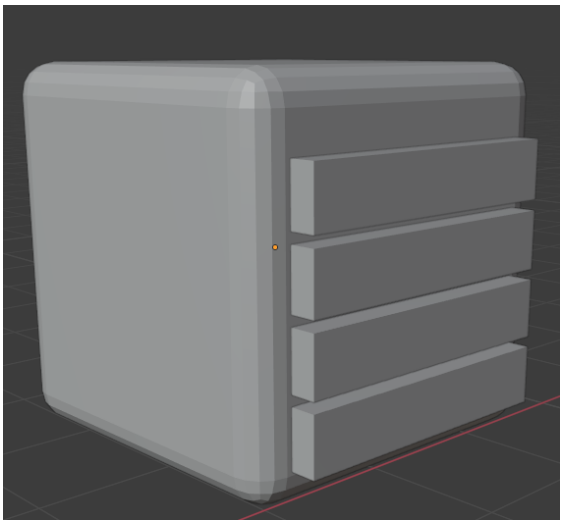


*Slika 1. Označeni redovi [autorski rad]*

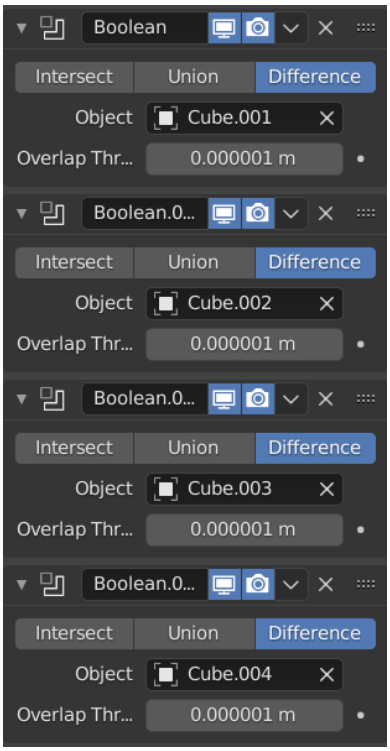


*Slika 2. Zaobljeni rubovi [autorski rad]*

Za još bolju glatkost, Blender ima ugrađeni mehanizam koji se koristi tako da se odabere željeni objekt te odabere sljedeća kombinacija: Object->Shade smooth. Ladice se rade korištenjem boolean modifikatora. Pomoću boolean modifikatora, Blender nalazi razlike u minimalno dva objekta te u ovom slučaju, ladice rade razliku u tijelu ormarića što rezultira praznom prostoru na ormariću gdje su postavljene ladice. Prvo je potrebno kreirati 4 ladice te ih postaviti na željenu poziciju glavnog tijela ormarića.

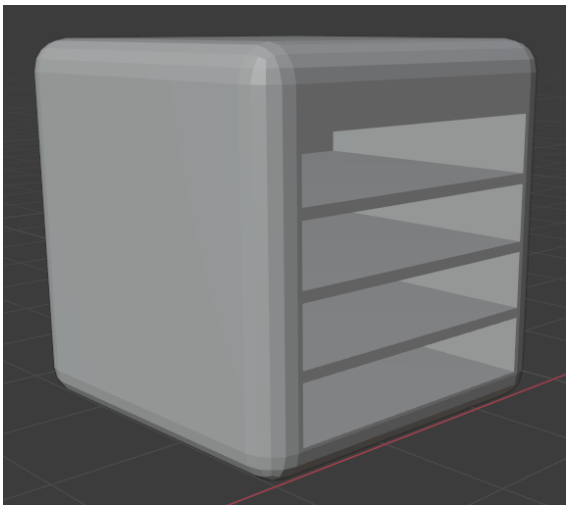


*Slika 3. Ormarić sa ladicama [autorski rad]*




*Slika 4. Boolean modifikator [autorski rad]*

Na slici 4. nalazi se prikaz boolean modifikatora koji je pridružen glavnom tijelu ormarića. Svaki boolean modifikator mora biti povezan sa specifičnom ladicom (ukupno 4). Slijedi slika rezultata boolean modifikatora.



*Slika 5. Rezultat boolean modifikatora [autorski rad]*

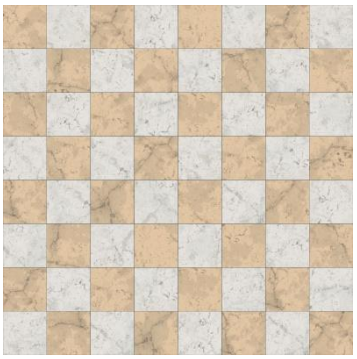
### 3.1.2.Teksturiranje modela

Teksturiranje je proces „oživljavanja“, u ovome slučaju objekata. To bi značilo davanje boja objektima kako bi poprimili svoj završni izgled. Teksture koje će se koristiti nalaze se na sljedećoj stranici, https://www.textures.com/

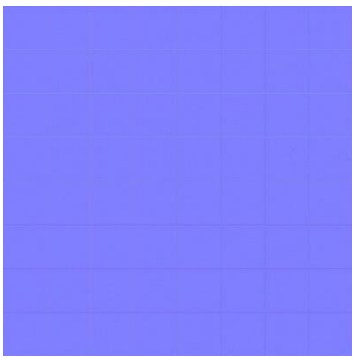
Preuzete teksture lijepit će se na već modelirane objekte. Zalijepljene teksture potrebno je prilagoditi objektima kako bi izgledale što realnije i prirodnije na objektima na koje se lijepe. Za neke objekte neće biti potrebno lijepiti strukture već manipulirati bojama i različitim postavkama kako bi boja, tekstura bila što prirodnija. Postavke pomoću kojih se manipulira bojama, teksturama zove se „Node editor“ tj. Uređivač čvorova. [1]

Slijedi opis teksturiranja pločica na podu. Tekstura pločica preuzeta je sa već ranije navedene stranice te ju je potrebno kroz uređivač čvorova i mapiranja dovesti do što realnijeg prikaza. Modelu pločica dodane su tri vrste tekstura, a to su:

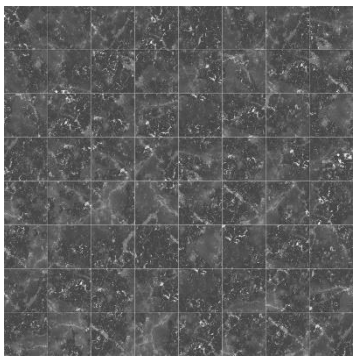
- Color Texture
    - Standardna boja materijala, boja koja se dobije slikanjem materijala s visoka
- Normal Texture
    - Idealna za prikaz 3D modela, daje prirodniji izgled reljefa
- Roughness Texture
    - Daje hrapavost i glatkost na specifičnim dijelovima materijala



*Slika 6. Color Texture [2]*

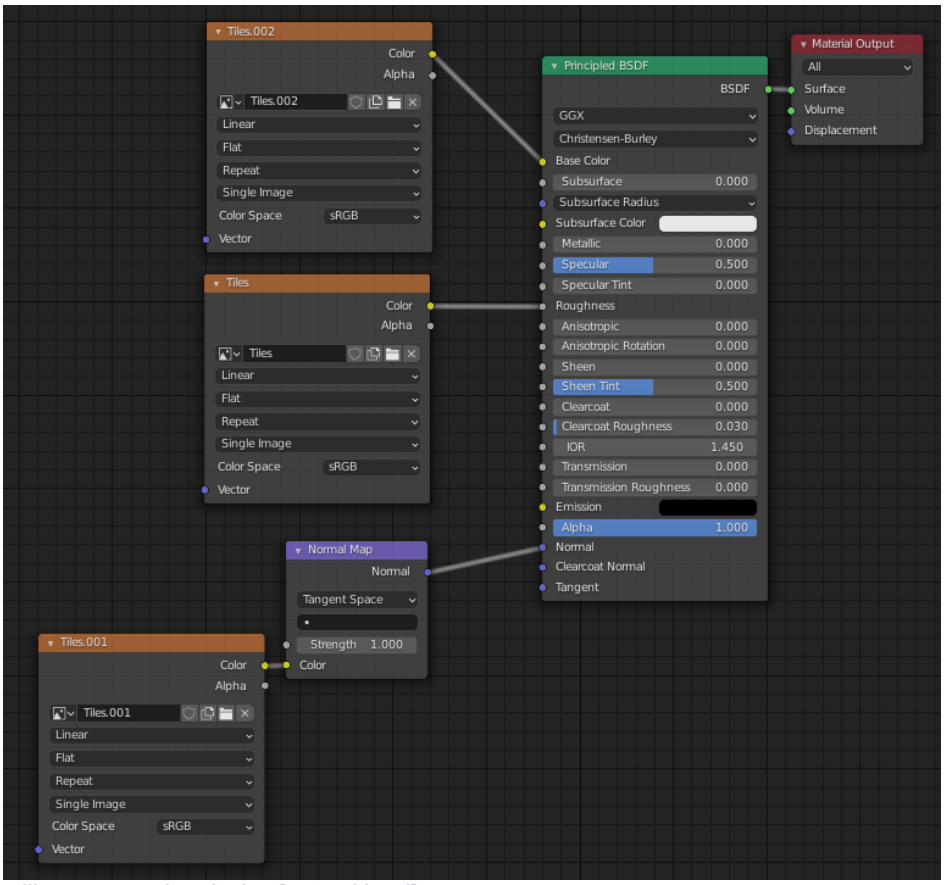


*Slika 7. Normal Texture [2]*



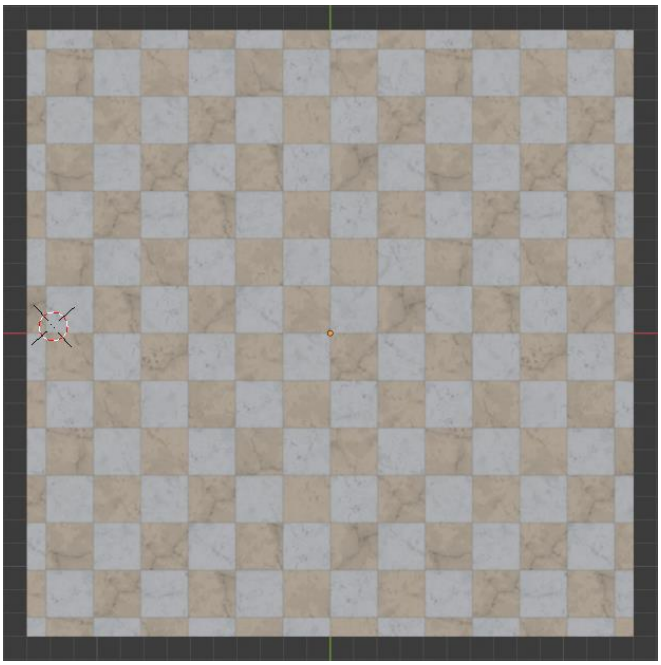
*Slika 8. Roughness Texture[2]*

Slijedi slika koja prikazuje uređivač čvorova teksture. Jasno se vidi da su pridružene Color, Normal i Roughness Texture.




*Slika 9. Postavke pločica [autorski rad]*

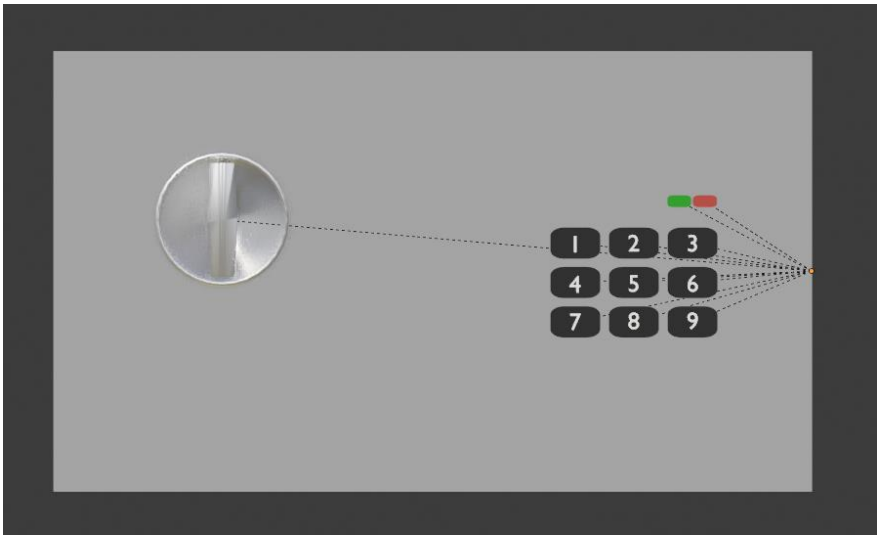
Teksturu koja koristi priložene slike potrebno je mapirati kako bi sjela na objekt u što prirodnijem i realnijem obliku. Slike ispod prikazuju pod nakon što je dodana tekstura pločica i odrađena postavka mapiranja.



*Slika 10. Pločice [autorski rad]*

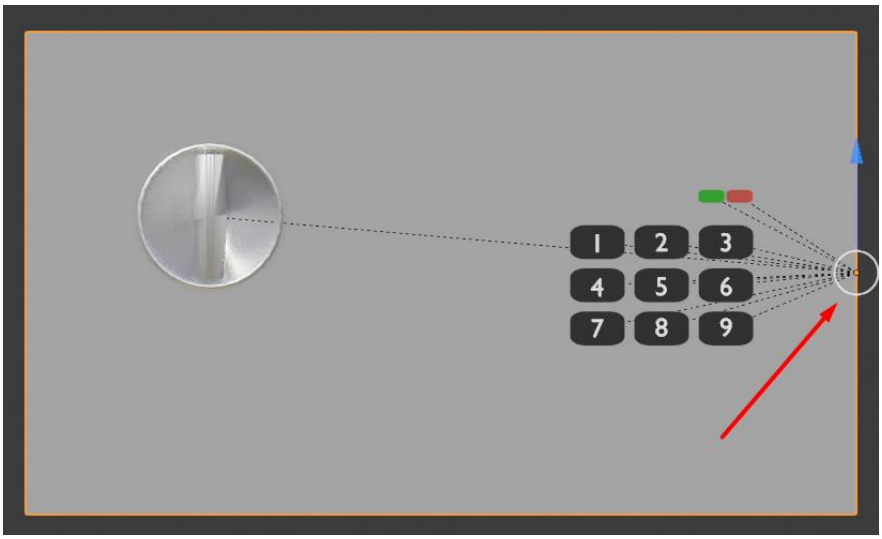
### 3.1.3. Animacija modela

Slijedi opis animiranja otvaranja sefa. Animiranje sefa sastoji se od animacije koja je povezana sa manipuliranjem rotacije objekta. Igrica isto tako sadrži i objekte koji imaju animacije povezane sa manipulacijom pozicije, npr. Ladice prilikom otvaranja i zatvaranja mijenjaju poziciju po jednoj od tri osi.




*Slika 11. Sef [autorski rad]*

Animacija otvaranja vrata sefa odvija se u dva slijeda. Prvo se drška sefa rotira za -90°, nakon čega se vrata sefa rotiraju za 50°. Prije samog animiranja, vrlo je bitno postaviti jezgru objekta (origin) jer ona je centar odvijanja tijekom manipuliranja objektom. Npr., centrala vrata ne može biti sama geometrijska centrala  već sami desni rub ili lijevi rub objekta (vertikalna sredina). Rečeno je jasnije vidljivo na sljedećoj slici.

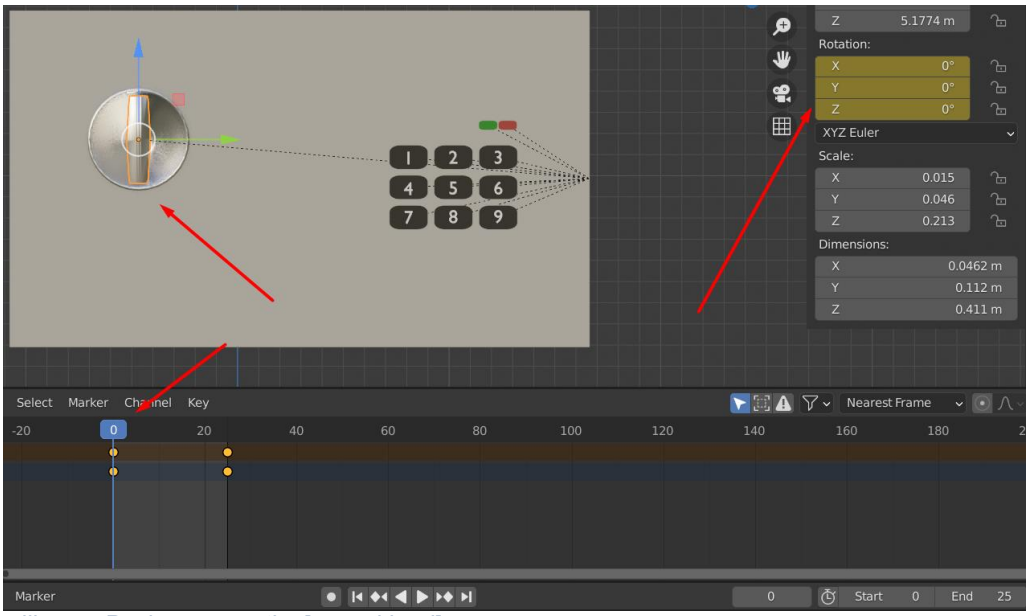



*Slika 12. Centrala objekta*

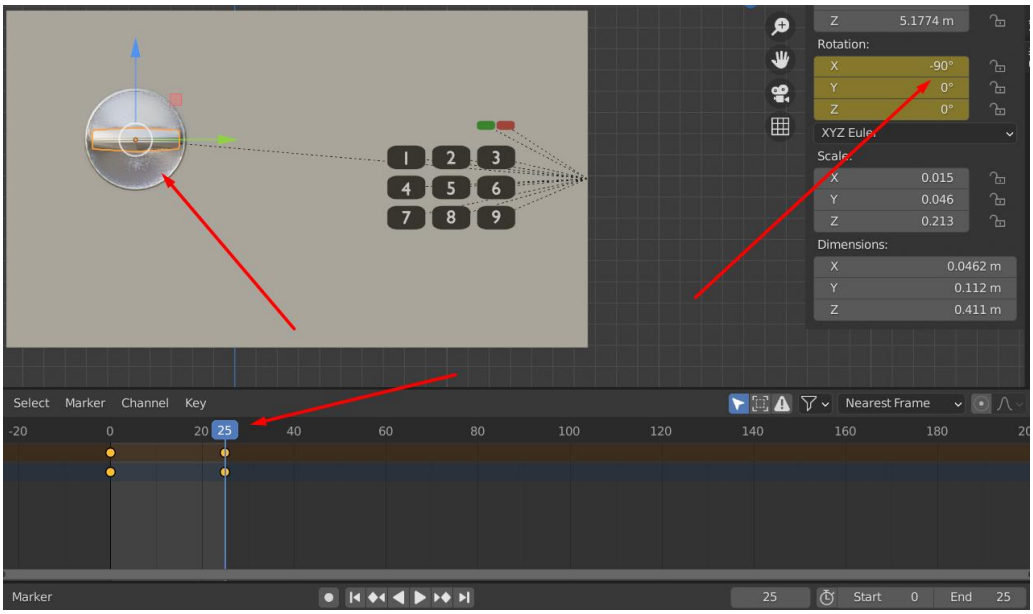
Za početak je potrebno postaviti interval vremenskog okvira odvijanja animacije (Frames). Ukoliko kasnije u Unityu primjetite da je animacija prebrza ili prespora, moguće je u samom Unity-u usporiti ili ubrzati animaciju.


*Slika 13. Vremenski okvir animacije*

Cilj je postaviti na 0. vremenskom okviru početnu rotaciju objekta i na zadnjem okviru, 25., završnu rotaciju objekta. Kao što je ranije rečeno, objekt ručke u 0. vremenskom okviru ima 0°. To se postavlja tako da se za ranije odabrani model ručke postavi ključni vremenski okvir (Key frame) rotacijskog tipa.  Isto tako, novi vremenski okvir ručke postavlja se na završnom 25. vremenskom okviru, ali ovog tipa rotacija se postavlja na -90°. Slijedi slika postavljanja rotacije na 0. vremenskom okviru i 25. Potrebno obratiti pozornost na vremensku crtu, položaj modela ručke i vrijednost rotacije po x osi.




*Slika 14. Ručka na početku [autorski rad]*




*Slika 15. Ručka na kraju [autorski rad]*

## 3.2. Poslovna logika

Ovo poglavlje sadrži objašnjenje koda koji gradi poslovnu logiku igrice. Poglavlje će biti strukturirano kroz slike koda gdje će svaka slika biti dodatno tekstualno pojašnjena.

### 3.2.1.FPController.cs

Skripta je dobivena povodom online tečaja koje je moguće kupiti na sljedećoj poveznici: https://www.udemy.com/course/procedural-maze-dungeon-generation/
Skripta sadrži algoritam za hodanje u prvom licu (First Person Character). Najvažnija je metoda FixedUpdate koja sadrži algoritam za hodanje pomoću tipkovnice i micanje kamere pomoću miša. Kod je objašnjen direktno u editor-u.

```
107     void FixedUpdate()
108     {
109         //Micanje miša lijevo desno. Sensitivity, koliko pokazivač miša brzo respondira prilikom micanja miša
110         float yRot = Input.GetAxis("Mouse X") * sensitivity;
111         float xRot = Input.GetAxis("Mouse Y") * sensitivity;
112 
113         //Središnji fokuser gore dolje
114         cameraRot *= Quaternion.Euler(-xRot, 0, 0);
115         //Središnji fokuser lijevo desno
116         characterRot *= Quaternion.Euler(0, yRot, 0);
117 
118         //Zabrani punu rotaciju po x osi središnjeg fokusera
119         cameraRot = ClampRotationAroundXAxis(cameraRot);
120 
121         this.transform.localRotation = characterRot;
122         cam.transform.localRotation = cameraRot;
123 
124         //Lijevo Desno na tipkovnici hodanje
125         x = Input.GetAxis("Horizontal") * speed;
126         //Gore Dolje na tipkovnici hodanje
127         z = Input.GetAxis("Vertical") * speed;
128 
129         //Nova pozicija igrača
130         transform.position += this.transform.forward * z + cam.transform.right * x;
131 
132         //Fokuser miša
133         UpdateCursorLock();
134     }
```

*Slika 16. FP Hodanje [autorski rad]*

## 3.2.2.SelectionManager.cs

Skripta predstavlja jezgru igre te joj je glavna funkcija prepoznati korisnikove želje prilikom manipuliranja objektima. Slijede kompleksniji dijelovi skripte. Prvo se pokreće Awake metode čija je glavna svrha pronaći objekte koji će se koristiti u ovoj skripti. Potrebno je također pronaći Canvas-e koji će se koristiti za prikaz Inventory-a te ih deaktivirati kako se na samom početku ne bi prikazali igraču (cilj je da se prikazuju preko tipke „I“).

```
61      private void Awake()
62      {
63          inventory = new Inventory();
64          uiInventory.SetInventory(inventory);
65
66          UiInventoryCanvas = GameObject.Find("UiInventory");
67          UiInventoryCanvas.SetActive(false);
68
69          UiInventoryRead = GameObject.Find("UiInventoryRead");
70          UiInventoryRead.SetActive(false);
71
72          objektSlike = GameObject.Find("Painting");
73
74          istocniZid = GameObject.Find("WallEast");
75      }
```

*Slika 17. SelectionManager.Awake [autorski rad]*

Metoda Start() poziva se nakon što se završi metoda Awake(). Ono što treba napomenuti u ovoj metodi su 81. i 82. linija. One služe za potencijalno pozicioniranje objekta kocki nakon što se dignu. FPC_ObjectHolder predstavlja prazni objekt koji predstavlja poziciju kocke nakon što ju glavni junak podigne. Između 84. i 97. linije nalazi se povezivanje svih 6 cilindara koji služe za provjeru kolizije kocke. Svaki cilindar ima svojstvo match tipa boolean koji provjerava je li tražena kocka spojena sa traženim cilindrom.

```
76      private void Start()
77      {
78          objectManipulation = GetComponent<ObjectManipulationText>();
79
80          //Dohvati roditelja. Služi za prijenos objekta. Razmak između igrača i objekta prilikom premještanja
81          trenutniRoditelj = GameObject.Find("FPC_ObjectHolder");
82          trenutniRoditeljManevriranje = trenutniRoditelj.GetComponent<Transform>();
83
84          #region Cylinders
85          GameObject c1 = GameObject.Find("Cylinder_1");
86          cylinder_1 = c1.GetComponent<Cylinder>();
87          GameObject c2 = GameObject.Find("Cylinder_2");
88          cylinder_2 = c2.GetComponent<Cylinder>();
89          GameObject c3 = GameObject.Find("Cylinder_3");
90          cylinder_3 = c3.GetComponent<Cylinder>();
91          GameObject c4 = GameObject.Find("Cylinder_4");
92          cylinder_4 = c4.GetComponent<Cylinder>();
93          GameObject c5 = GameObject.Find("Cylinder_5");
94          cylinder_5 = c5.GetComponent<Cylinder>();
95          GameObject c6 = GameObject.Find("Cylinder_6");
96          cylinder_6 = c6.GetComponent<Cylinder>();
97          #endregion
98      }
```

*Slika 18. SelectionManager.Start [autorski rad]*

Slijedi prolazak kroz najvažniju metodu ove igre, a to je metoda Update. Metoda je „poveća“ tako da će biti rascjepkana na više dijelova. Sljedeći izrezak koda prikazuje provjeru je li došlo do pogotka između kocki i cilindara. Svaka od šest kocki ima svoj pripadajući cilindar. Ukoliko svi cilindri imaju boolean svojstvo match==true, dešava se akcija animacije otvaranja ormara

sa knjigama te vrata koja se nalaze iza tog ormara postaju aktivna za označavanje i manipulaciju između igrača i vrata.

```
524     private void CubeCylinderMatch()
525     {
526         if (bookShelfOpened == false)
527         {
528             if (cylinder_1.match &&
529                 cylinder_2.match &&
530                 cylinder_3.match &&
531                 cylinder_4.match &&
532                 cylinder_5.match &&
533                 cylinder_6.match)
534             {
535                 Debug.Log("FULL MATCH");
536                 bookShelfOpened = true;
537 
538                 AnimController bsa = GameObject.Find("Bookshelf").GetComponent<AnimController>();
539                 bsa.StartBookShelfAnimation();
540                 GameObject d = GameObject.Find("Door");
541                 d.tag = "ObjectSelectable_Door";
542             }
543         }
544     }
```

*Slika 19. CubeCyilinderMatch [autorski rad]*

Sljedeći dio koda služi za bacanje kocke. Prvo se provjerava je li kocka dignuta, ako je, slijedi spuštanje iste (ukoliko igrač to želi i pritisne „e“). Prilikom ispuštanja kocke, potrebno je aktivirati BoxCollider (služi za provjeru kolizije sa ostalim objektima) koji se isključio nakon podizanja. Razlog isključenja BoxCollidera tijekom nošenja je izbjegavanje kolizije sa ostalim objektima te moguće neželjeno ispadanje kocke prilikom nošenja. Kako bi kocka pala na pod, potrebno je uključiti gravitaciju pomoću Rigidbody komponente. 148. linija koda pokreće korutinu koja služi za odgađanje akcije, točnije, cilj je da nakon jedne sekunde, objekt kocke ponovno poprimi tag „Selectable“ kako bi se ponovno mogao uzeti ukoliko je pogrešno stavljen.

```
135         if (objectRaised == true)
136         {
137             if (Input.GetKeyDown("e"))
138             {
139                 raisedObject.tag = "Untagged";
140                 raisedObject.GetComponent<BoxCollider>().enabled = true;
141                 raisedObject.GetComponent<Rigidbody>().useGravity = true;
142                 raisedObject.transform.localEulerAngles = new Vector3(0, 0, 0);
143                 raisedObject.transform.parent = null;
144                 objectRaised = false;
145                 //Pričekaj da kocka padne kako bi ponovno mogla biti
146                 //selektirana jer ne želimo da je korisniku tijekom
147                 //nošenja stalno highlight-ana kocka
148                 StartCoroutine("WaitCubeDrop");
149             }
150         }
```

*Slika 20. Ispuštanje kocke [autorski rad]*

```
99      public IEnumerator WaitCubeDrop()
100     {
101         yield return new WaitForSeconds(1f);
102         raisedObject.tag = "ObjectSelectable_Cube";
103     }
```

*Slika 21. Čekanje prilikom spuštanja kocke [autorski rad]*

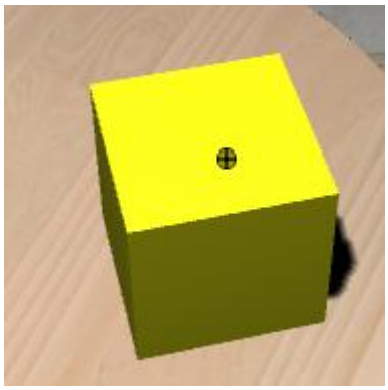
Igrač može provjeriti prikupljene objekte pritiskom na tipku „I“. Pritiskom na tipku „I“ korisniku se aktivira Canvas na kojemu se nalazi Inventory te objašnjenje korištenja.

```
152             //Korištenje Inventory-a
153             if (Input.GetKeyDown("i"))
154             {
155                 if (UiInventoryCanvas.activeInHierarchy == true)
156                 {
157                     UiInventoryCanvas.SetActive(false);
158                 }
159                 else
160                 {
161                     UiInventoryCanvas.SetActive(true);
162                     objectManipulation.currentManipulationText.text = "Select Inventory Item " +
163                         "By Entering Required ID";
164                 }
165             }
```

*Slika 22. Inventory [autorski rad]*

Objekt koji se može selektirati i koji je fokusiran poprima material žute boje. Slijedi primjer.

*Slika 23. Objekt prije fokusiranja [autorski rad]*



*Slika 24. Objekt poslije fokusiranja [autorski rad]*

Kako bi objekt znao koji teksturu mora vratiti nakon micanja fokusa, svaki objekt mora znati koje su njegove teksture tj. njegovi materiali. Neki objekti imaju više materiala tako da su materiali pojedinog objekta pohranjeni u polje materiala (184. linija). Sljedeći kod prikazuje algoritam vraćanja tekstura objekta u prvotno stanje nakon što pojedini objekt izgubi fokus igrača.

```
179             if (_selection != null)
180             {
181                 var selectionRenderer = _selection.GetComponent<Renderer>();
182                 SelectableObject selectedObject = _selection.GetComponent<SelectableObject>();
183
184                 Material[] materials = new Material[selectionRenderer.materials.Length];
185                 for (int i = 0; i < selectionRenderer.materials.Length; i++)
186                 {
187                     materials[i] = selectedObject.GetDefaultMaterials()[i];
188                 }
189                 selectionRenderer.sharedMaterials = materials;
190
191                 _selection = null;
192             }
```

*Slika 25. Micanje fokusa [autorski rad]*

Fokusiranje objekta je implementirano pomoću RaycastHit strukture. 195. linija služi za prikaz ciljnika koji je pozicioniran točno na vertikalnoj i horizontalnoj sredini. Važno je napomenuti da RaycastHit vidi samo objekte koji imaju BoxCollider. U 202. liniji definirana je željena udaljenost od objekta prilikom fokusiranja.

```
194             //Ciljnik za selektiranje točno na vertikalnoj i horizontalnoj sredini
195             var ray = Camera.main.ViewportPointToRay(new Vector2(0.5f, 0.5f));
196             //var ray = Camera.main.ScreenPointToRay(Input.mousePosition);
197             RaycastHit hit;
198             //Ukoliko je fokusiran objekt koji ima box collider
199             if (Physics.Raycast(ray, out hit))
200             {
201                 //Udaljenost od fokusiranog objekta
202                 if (hit.distance <= 6)...
474                 else...
479             }
```

*Slika 26. Raycast [autorski rad]*

Ranije je opisano vraćanje originalne teksture objekta nakon micanja fokusa, slijedi kod koji postavlja žutu teksturu objektu nakon što je objekt fokusiran. 225. linija prikazuje polje gdje svaki material fokusiranog objekta poprima žuti material (u slučaju da selektirani objekt ima više tekstura).

```
218             //Fokusirani objekt dobiva žuti material
219             var selectionRenderer = selection.GetComponent<Renderer>();
220             if (selectionRenderer != null)
221             {
222                 Material[] materials = new Material[selectionRenderer.materials.Length];
223                 for (int i = 0; i < selectionRenderer.materials.Length; i++)
224                 {
225                     materials[i] = highlightMaterial;
226                 }
227                 selectionRenderer.sharedMaterials = materials;
228             }
```

*Slika 27. Highlight [autorski rad]*

Svaki selektirani objekt se može kontrolirati. Da bi se objekt mogao kontrolirati, potrebno je pritisnuti tipku „e“. Isto tako, svaki selektirani objekt ima svoj znak (tag) pomoću kojeg se zna što učiniti sa istim, koju animaciju pokrenuti, koju akciju pokrenuti i sl.

ObjectSelectable_Inventory
ObjectSelectable_Drawer
ObjectSelectable_Door
ObjectSelectable_Painting
ObjectSelectable_SafeButton
ObjectSelectable_Switch
ObjectSelectable_Cube
ObjectSelectable_Readable

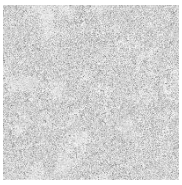
*Slika 28. Tag-ovi objekata za prikupljanje [autorski rad]*

Ukoliko su vrata fokusirana, pritisnuta je tipka „e“ te su vrata otključana (DoorUnLocked: Boolean), pokreće se animacija (Animacije će biti objašnjene kasnije u skripti AnimController.cs) otvaranja vrata.

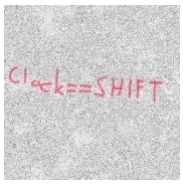
```
237             case "ObjectSelectable_Door":
238                 {
239                     //Provjera jesu li vrata otključana
240                     //Ako jesu, pokreni animaciju otvaranja
241                     if (DoorUnLocked)
242                     {
243                         AnimController ac1 = GameObject.Find("WallNorth").GetComponent<AnimController>();
244                         ac1.StartDoorAnimation();
245                     }
246                     else
247                     {
248                         Debug.Log("Door is locked.");
249                     }
250                     break;
251                 }
```

*Slika 29. ObjectSelectable_Door [autorski rad]*

Objekt prekidača ima dvije animacije, paljenje i gašenje. Prekidač pali UV svjetlo (283. linija) koje prikazuje skrivenu poruku (280. linija) na istočnom zidu. Istočni zid ima dvije teksture, jedna tekstura je obična, a druga tekstura je ista kao i prva uz dodatak poruke.



*Slika 30. Tekstura bez poruke [3]*




*Slika 31. Tekstura sa porukom [autorski rad]*

```
252             case "ObjectSelectable_Switch":
253                 {
254                     AnimController ac = GameObject.Find("Switch").GetComponent<AnimController>();
255                     ac.StartSwitchAnimation(this.switchTurnedOn);
256                     if (this.switchTurnedOn)
257                     {
258                         this.switchTurnedOn = false;
259                         var wallRenderer = this.istocniZid.GetComponent<Renderer>();
260                         WallWithHidenInfo wall = this.istocniZid.GetComponent<WallWithHidenInfo>();
261                         if (wallRenderer != null)
262                         {
263                             Material[] materials = new Material[wallRenderer.materials.Length];
264                             materials[0] = wall.GetDefaultMaterial();
265                             materials[1] = wall.GetDefaultMaterial();
266                             wallRenderer.sharedMaterials = materials;
267                             Light sl = GameObject.Find("UvLamp").GetComponent<Light>();
268                             sl.intensity = 0;
269                         }
270                     }
271                     else
272                     {
273                         this.switchTurnedOn = true;
274                         var wallRenderer = this.istocniZid.GetComponent<Renderer>();
275                         WallWithHidenInfo wall = this.istocniZid.GetComponent<WallWithHidenInfo>();
276                         if (wallRenderer != null)
277                         {
278                             Material[] materials = new Material[wallRenderer.materials.Length];
279                             materials[0] = wall.GetDefaultMaterial();
280                             materials[1] = wall.hiddenMessageMaterial;
281                             wallRenderer.sharedMaterials = materials;
282                             Light sl = GameObject.Find("UvLamp").GetComponent<Light>();
283                             sl.intensity = 0.5f;
284                         }
285                     }
286                     break;
```

*Slika 32.ObjectSelectable_Switch [autorski rad]*

Postoje dva objekta koji imaju tag „ObjectSelectable_Drawer“. Jedan ormarić je zaobljen, a drugi je pravokutan („Cube“, 297. linija). Potrebno je provjeriti o kojem je ormariću riječ kako bi se znalo koju animaciju pokrenuti. Pravokutni ormarić je zaključan (307. linija) te se animacija pokreće tek kad je javno svojstvo CurvedDrawerUnLocked==true. Kasnije će biti prikazana metoda koja služi za otključavanje objekata. 302. i 303. linija prikazuju paljenje svjetla u ladici pravokutne ladice. Razlog tome je što se u ladici nalazi objekt kojeg je potrebno osvijetliti kako bi igrač bolje vidio što se nalazi unutar iste.

```
295                 case "ObjectSelectable_Drawer":
296                     {
297                         if (hit.transform.name.Contains("Cube"))
298                         {
299                             //Upali svjetlo u ladici da se bolje vidi papirić koji se tamo nalazi
300                             AnimController ac = GameObject.Find("DrawerCube").GetComponent<AnimController>();
301                             ac.StartDrawerAnimation(hit.transform.name);
302                             Light sl = GameObject.Find("DrawerCubeChildLight").GetComponent<Light>();
303                             sl.intensity = 1.2f;
304                             break;
305                         }
306                         //Provjera je li ormarić zaključan
307                         if (CurvedDrawerUnLocked)
308                         {
309                             AnimController ac = GameObject.Find("CurvedDrawer").GetComponent<AnimController>();
310                             ac.StartDrawerAnimation(hit.transform.name);
311                         }
312                         else
313                         {
314                             Debug.Log("Drawer is locked.");
315                         }
316                         break;
317                     }
```

*Slika 33. ObjectSelectable_Drawer [autorski rad]*

Sljedeći dio koda služi za dodavanje objekta u Inventory. 321. linija dodajte objekt u listu objekata. Poziva se metoda sa sljedećim argumentima:

- itemType – tip objekta koji služi za povezivanje odgovarajuće sličice prilikom prikaza Inventorya
- description – opis objekta
- inventoryImage – koriste objekti za čitanje (knjiga, bilješke)
- id – najvažniji kod ključeva kako bi se znalo koji ključ otvara koja vrata

Nakon dodavanja objekta u Inventory, potrebno ga je izbrisati iz scene (linija 328). Nakon brisanje je potrebno osvježiti Inventory kako bi se korisniku prikazao najnovije dodani objekt, a to se postiže pomoću EventHandler-a koji je objašnjen u naslovu Inventory.cs.

```
334                 case "ObjectSelectable_Inventory":
335                     {
336                         inventory.AddItem(new Item {
337                             itemType = hit.transform.GetComponent<ItemWorld>().itemType,
338                             description = this.inventoryItemId.ToString(),
339                             inventoryImage = hit.transform.GetComponent<ItemWorld>().inventoryImage,
340                             id = hit.transform.GetComponent<ItemWorld>().id });
341                         //Nakon što je objekt prikupljen potrebno ga je uništiti iz scene
342                         Destroy(hit.transform.gameObject);
343                         //Osvježi inventory kako bi prikazao novo prikupljeni objekt
344                         //uiInventory.RefreshInventoryItems();
345                         this.inventoryItemId++;
346                         //Ugasi svjetlo u ladici
347                         if (hit.transform.name == "SvahiliNote")
348                         {
349                             Light sl = GameObject.Find("DrawerCubeChildLight").GetComponent<Light>();
350                             sl.intensity = 0;
351                         }
352                         break;
353                     }
```

*Slika 34. ObjectSelectable_Inventory [autorski rad]*

Objekt slike ima dvije funkcije. Prva je da sakrije što se nalazi iza nje, a druga je da padne na pod nakon što igrač to zaželi. 343. linija uključuje gravitacija te slika automatski pada na pod, a 345. linija koda stavlja novi znak (tag) objektu. Razlog tome je da se igraču onemogući nakon bacanja objekta slike na pod ponovno dizanje tj. korištenje tj. selektiranje objekta (nepotrebno).

```
340                 case "ObjectSelectable_Painting":
341                     {
342                         //Baci sliku na pod
343                         objektSlike.GetComponent<Rigidbody>().useGravity = true;
344                         objektSlike.transform.parent = null;
345                         objektSlike.tag = "Untagged";
346                         break;
347                     }
```

*Slika 35.ObjectSelectable_Painting [autorski rad]*

Sljedeći dio koda odnosi se na korištenje kocki koje su posložene po sobi. Moguće akciju su podizanje i spuštanje. Prvo je potrebno dohvatiti objekt kocke (350. linija). Zatim se provjerava je li kocka podignuta, ako nije, dopusti podizanje iste. Razlog ove provjere je sprječavanje igraču da digne dvije kocke u isto vrijeme. U istom trenutku moguće je podignuti samo jednu kocku. Nakon što se kocka podigne, potrebno je isključiti detektiranje kolizije za tu kocku kako bi se spriječilo zapinjanje sa ostalim objektima koji imaju svojstvo provjere kolizije. Ranije je spomenut objekt trenutniRoditelj (FPC_ObjectHolder) koji ima funkciju pozicije nošenog objekta. 359. linija postavlja kocku na poziciju trenutniRoditelj. 370. linija pokreće korutinu čiji je zadatak pričekati 0.1s prilikom podizanja objekta kocke te nakon toga ugasiti svojstvo provjere kolizije. Razlog tome je što cilindar za detektiranje spuštene kocke ne prepoznaje da je objekt kocke maknut sa istog ukoliko kocka nema svojstvo kolizije.

```
348                 case "ObjectSelectable_Cube":
349                     {
350                         GameObject selectedCube = GameObject.Find(hit.transform.name);
351                         //Ako kocka nije podignut, dopusti podizanje
352                         //Sprijeci podizanje kocke ukoliko je već podignuta
353                         if (objectRaised == false)
354                         {
355                             //Ugasi detektiranje kolizije kako ne bi zapinjao sa kockom prilikom nošenja
356                             selectedCube.GetComponent<Rigidbody>().useGravity = false;
357                             selectedCube.GetComponent<Rigidbody>().detectCollisions = true;
358                             selectedCube.transform.parent = trenutniRoditelj.transform;
359                             selectedCube.transform.position = trenutniRoditeljManevriranje.transform.position;
360 
361                             selectedCube.transform.localEulerAngles = new Vector3(0, 0, 0);
362                             //selectedCube.GetComponent<BoxCollider>().enabled = false;
363                             objectRaised = true;
364                             raisedObject = selectedCube;
365 
366                             //Pričekaj da se kocka digne pa tek onda ugasi collider
367                             //Razlog tome je što cilindar za detektiranje kolizije
368                             //ne prepoznaje da je objekt maknut sa istog ukoliko ne postoji collider
369                             //Zato se čeka 0.1s da se kocka digne pa se tek onda collider ugasi
370                             StartCoroutine("WaitCubeRise");
371                         }
372                         break;
373                     }
```

*Slika 36. ObjectSelectable_Cube [autorski rad]*

ObjectSelectable_Readable je oznaka (tag) koja označava da objekt ima funkciju čitanja. Prvo je potrebno dohvatiti objekt slike (Raw Image), 377. linija.

378. linija dohvaća skriptu koja sadrži svojstvo inventoryImage tipa Texture u kojoj se nalazi slika koju je potrebno prikazati korisniku. Linija 379. povezuje Raw Image objekt sa dohvaćenom slikom fokusiranog objekta.

```
374                 case "ObjectSelectable_Readable":
375                     {
376                         UiInventoryRead.SetActive(true);
377                         Transform rawImage = UiInventoryRead.transform.GetChild(0).GetChild(0);
378                         ItemWorld iw = hit.transform.GetComponent<ItemWorld>();
379                         rawImage.GetComponent<RawImage>().texture = iw.inventoryImage;
380                         break;
381                     }
```

*Slika 37. ObjectSelectable_Readable [autorski rad]*

U ranijem naslovu prikazan je model sefa koji na sebi ima devet tipki. Komunikacija sa tipkama nije implementirana klikom na tipku „e“ već lijevim klikom miša. Prije pokretanja animacije otvaranja sefa, potrebno je unesti ispravnu šifru. Sef ima skriptu koja provjerava ispravnost šifre. Svaki put kad se pritisne tipka, poziva se metoda CheckPassword i prenosi se trenutno unesena šifra. Nakon unosa četiri znaka lozinke, lozinka se resetira (šifra je dužine četiri znaka).

```
9   public static class Safe
10  {
11      public static string Password = "3728";
12
        1 reference
13      public static bool CheckPassword(string insertedPassword)
14      {
15          bool correct = false;
16
17          if (insertedPassword == Password)
18          {
19              correct = true;
20          }
21          return correct;
22      }
23  }
```

*Slika 38. CheckPassword [autorski rad]*

Postoje objekti koji su zaključani te se ne mogu koristiti prije korištenja specifičnog ključa. Svaki ključ ima dodijeljeni ID te je mapiran sa objektom kojeg je potrebno otključati.


*Slika 39. ID ključa [autorski rad]*

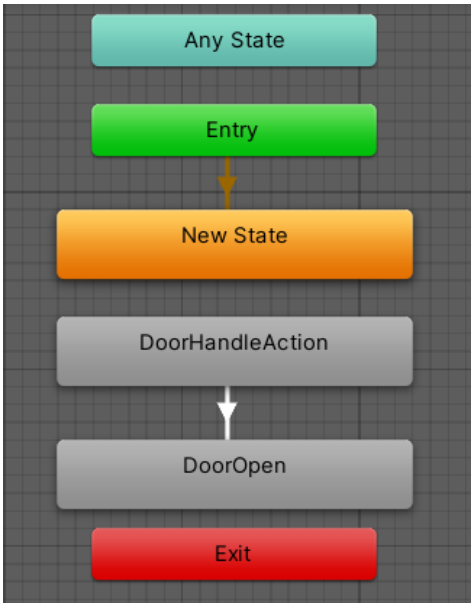
Prvo slijedi provjera ima li označeni objekt (npr. Vrata, ormarić i sl.) ID jednak ključu koji se želi koristiti. Ako nema, ispiši korisniku grešku. Ako je ključ ispravan, nastavi poslovnu logiku koja u ovom slučaju postavlja željeni objekt u stanje „otključan“, briše ključ iz Inventory-a (jer je ključ iskorišten) te javlja korisniku da je objekt uspješno otključan.

```
566     if (selectedItem.id == 11)
567     {
568         this.CurvedDrawerUnLocked = true;
569         SelectableObject selectedObject = GameObject.Find("CurvedDrawer").GetComponent<SelectableObject>();
570         selectedObject.locked = false;
571         inventory.RemoveItem(selectedItem);
572         objectManipulation.ShowFloatingText(_selection.name, _selection.tag);
573     }
574     else
575     {
576         StartCoroutine(objectManipulation.ShowWarningText("Wrong Key!"));
577     }
```

*Slika 40. Provjera ključa [autorski rad]*

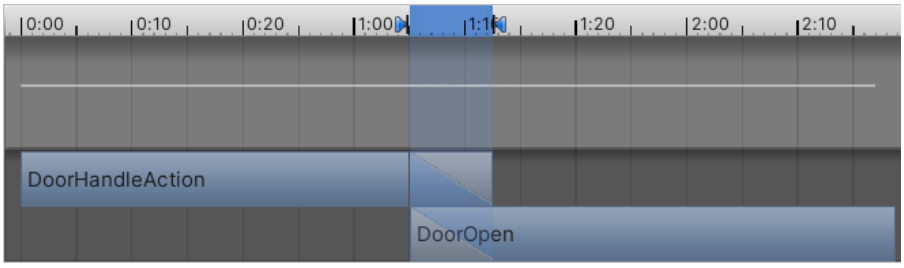
### 3.2.3. AnimController.cs

AnimController je skripta koja sadrži logiku za upravljanje i pokretanje animacija (koje su napravljene u Blender-u) na objektima. Svaki objekt koji sadrži animaciju, mora koristiti ovu skriptu. Prije opisa programskog koda navedene skripte, potrebno je navesti koje sve komponente su obavezne prije pokretanja animacije. Prvo je potrebno kreirati Animator Controller. Animator Controller je Unity-ev mehanizam koji omogućuje slaganje i održavanje skupa animacijskih isječaka te povezivanje prijelaza animacije za željeni objekt. Slijedi prikaz Animator Controller-a koji sadrži animacije povezane sa vratima. Animator Controller prikazuje tranziciju između dvije animacije, DoorHandleAction i DoorOpen što bi značilo da se prvo pokreće animacija otvaranja kvake i nakon završetka se pokreće animacija otvaranja vrata. Potrebno je povezati uvedenu animaciju objekta (Iz Blendera) sa željenim stanjem (Mapiranje DoorHandleAction sa kreiranom animacijom).



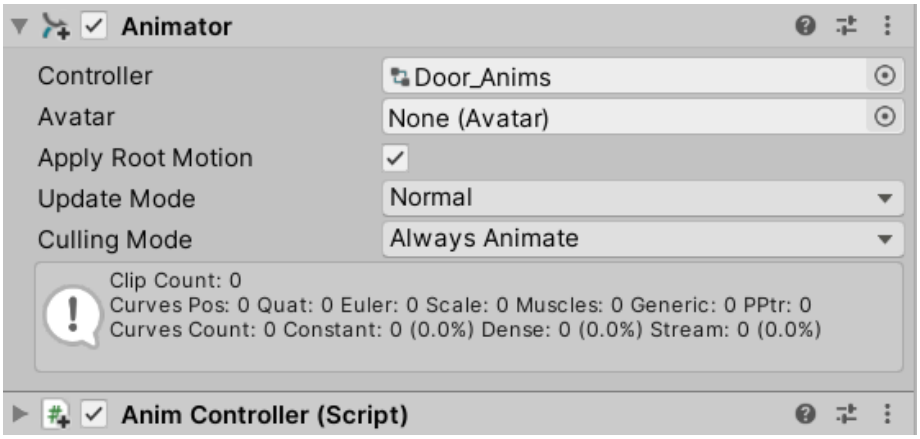

*Slika 41. Animator Controller [autorski rad]*

Animacija otvaranja vrata pokreće se malo prije nego što završi animacija otvaranja kvake (prirodniji izgled animacije).




*Slika 42. Prijelaz animacije [autorski rad]*

Nakon što je kreiran Animator Controller, potrebno ga je postaviti željenom objektu (Isto kao što je povezana i skripta AnimController.cs). Apply Root Motion je označen jer objekt mijenja poziciju tijekom animacije (To ne želimo).




*Slika 43. Animacija na objektu [autorski rad]*

Iz skripte AnimController.cs potrebno je samo pokazati kako se određena animacija poziva. Prvo je potrebno povezati komponentu Animatora sa objektom koji treba pokrenuti animaciju. Ta se operacija odrađuje u metodi Start. Metoda StartDoorAnimation pokreće željenu animaciju tako da kao argument pozivne metode Play priloži naziv animacije koja je kreirana u Animator Controlleru.

```
11      private Animator anim;
        Unity Message | 0 references
12      void Start()
13      {
14          anim = GetComponent<Animator>();
15          if (anim != null)
16          {
17              Debug.Log("Anim info: " + anim.name);
18          }
19      }
```

*Slika 44. Dohvaćanje Animatora [autorski rad]*

```
67      public void StartBookShelfAnimation() {
68          anim.Play("BookShelfOpen");
69          Debug.Log("BookShelf Animation");
70      }
```

*Slika 45.Pokretanje animacije [autorski rad]*

## 3.2.4.EndGame.cs

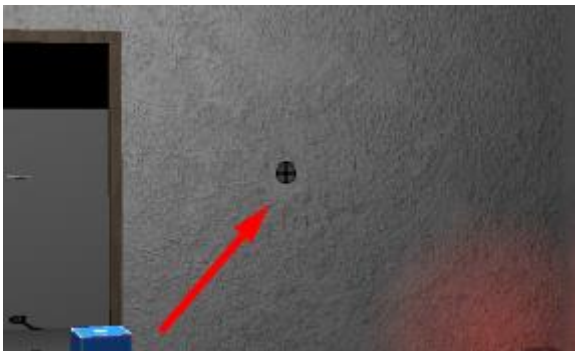
Riječ je o skripti koja se izvršava na kraju tj. u trenutku kad igrač završi sve zagonetke i izađe iz sobe. Skripta je pridružena objektu zida koji predstavlja granicu koja označava kraj igre.


*Slika 46. Granica kraja igre [autorski rad]*

Kad se igrač sudari sa granicom izvršava se metoda OnTriggerEnter koja se aktivira kod kolizije sa drugim objektom. Poziv ove metode znači kraj razine. Prvo je potrebno zaustaviti vrijeme. Vrijeme se stopira pozivom metode StopTimer koja se nalazi u klasi Stopwatch.cs. Zatim se mora onemogućiti glavnom junaku igre daljnje kretanje. Linija 38. dohvaća glavnog junaka igre, a linija 40. onemogućuje daljnje korištenje glavnog junaka. Zatim slijedi gašenje skripte PlayerControls jer ona crta ciljnik na vertikalnoj i horizontalnoj sredini. Vrijeme, naziv sobe i naziv igrača potrebno je spremiti pomoću metode StoreData koja će kasnije biti objašnjena. Rezultati se prikazuju aktiviranjem endGamePanel-a u liniji 47. Kursor miša je tijekom igre ugašen jer korisnik ima ciljnik (PlayerControls.cs) tako da je potrebno korisniku nakon završetka igre, prikazati kursor miša kako bi mogao odabrati želi li ponoviti igru ili se vratiti u glavni izbornik. Korisniku se na panelu prikazuje vrijeme potrebno za izvršavanje sobe i trenutna pozicija uspoređujući sa ostalim vremenima (Top 5 rezultata). Tu akciju odrađuje metoda GetRanks koja će kasnije biti razrađena.

```
33  private void OnTriggerEnter(Collider other)
34  {
35      Stopwatch bsa = GameObject.Find("UiStopwatch").GetComponent<Stopwatch>();
36      bsa.StopTimer();
37
38      FPController D = player.GetComponent<FPController>();
39      D.gameEnded = true;
40      D.enabled = false;
41
42      PlayerControls pc = mainCamera.GetComponent<PlayerControls>();
43      pc.enabled = false;
44
45      StoreData(bsa.currentTime);
46
47      endGamePanel.SetActive(true);
48
49      Cursor.visible = true;
50      Cursor.lockState = CursorLockMode.None;
51
52      Text timeText = GameObject.Find("UiTimeTxt").GetComponent<Text>();
53
54      TimeSpan time = TimeSpan.FromSeconds(bsa.currentTime);
55      timeText.text = "Time: " + time.ToString(@"mm\:ss\:fff");
56
57      GetRanks(bsa.currentTime);
58  }
```

*Slika 47. OnTriggerEnter [autorski rad]*



*Slika 48. Ciljnik [autorski rad]*


*Slika 49. Panel kraj igre [autorski rad]*

Za pohranu podataka koristi se jednostavni mehanizam Unity-a koji je idealan za manje, lokalne igre kao što je i ova. PlayerPrefs je klasa koja pohranjuje preferencije igrača između sesija igre. Može pohraniti string, float i integer vrijednosti u registar platforme korisnika. Metoda StoreData pohranjuje string tip objekta. Sprema se JSON koji se kreira pomoću Data.cs klase koja sadrži tri svojstva: RoomName, PlayerName i Time. Isto tako napravljena je klasa MemoryData koja sadrži listu objekta klase Data (Jedna instanca klase Data predstavlja jedno vrijeme tj. jednu igru) te joj je primarna svrha serializiranje u JSON i deserializiranje iz JSON-a. StoreData prvo mora dohvatiti sve prijašnje rezultate koji su upakirani u JSON te deserializirati iste. Zatim se dodaje novi rezultat na kraj liste i sprema se nova lista preko stare.

```
214     public void StoreData(float time)
215     {
216         string loadedResultsJson = PlayerPrefs.GetString("Result");
217         MemoryData loadedMemory = new MemoryData();
218         if (!String.IsNullOrEmpty(loadedResultsJson))
219         {
220             loadedMemory = JsonUtility.FromJson<MemoryData>(loadedResultsJson);
221         }
222
223         string roomName = SceneManager.GetActiveScene().name;
224         string playerName = PlayerPrefs.GetString("PlayerName");
225
226         Data result = new Data();
227         result.PlayerName = playerName;
228         result.Time = time;
229         result.RoomName = roomName;
230
231         loadedMemory.resultList.Add(result);
232
233         string resultsJson = JsonUtility.ToJson(loadedMemory);
234
235         PlayerPrefs.SetString("Result", resultsJson);
236         PlayerPrefs.Save();
237     }
```

*Slika 50. StoreData [autorski rad]*

Metoda GetRanks ima sljedeće zadatke kronološki poredane:

- Deserializirati sve pohranjene rezultate
- Rezultate poredati od najboljeg do najlošijeg
    - OrderBy
- Dohvatiti Top 5 rezultata
- Ispisati te rezultate na panel
- Trenutni rezultat igre smješten je u taj poredak te ima drugačiju boju fonta (plava) kako bi igrač znao koji je po redu vremenski

## 3.2.5.LaunchManager.cs

Skripta koja predstavlja logiku scene glavnog menija. Sadrži metode za dohvaćanje rezultata, dohvaćanje soba, mijenjanje soba, brisanje memorije i učitavanje scene odabrane igre. Metoda start dohvaća panel koji prikazuje rezultate te sakriva isti (Panel se prikazuje tek na pritisak specifičnog gumba), dohvaća sobe koje su dostupne za igru i dohvaća ime zadnjeg registriranog igrača (34. linija)

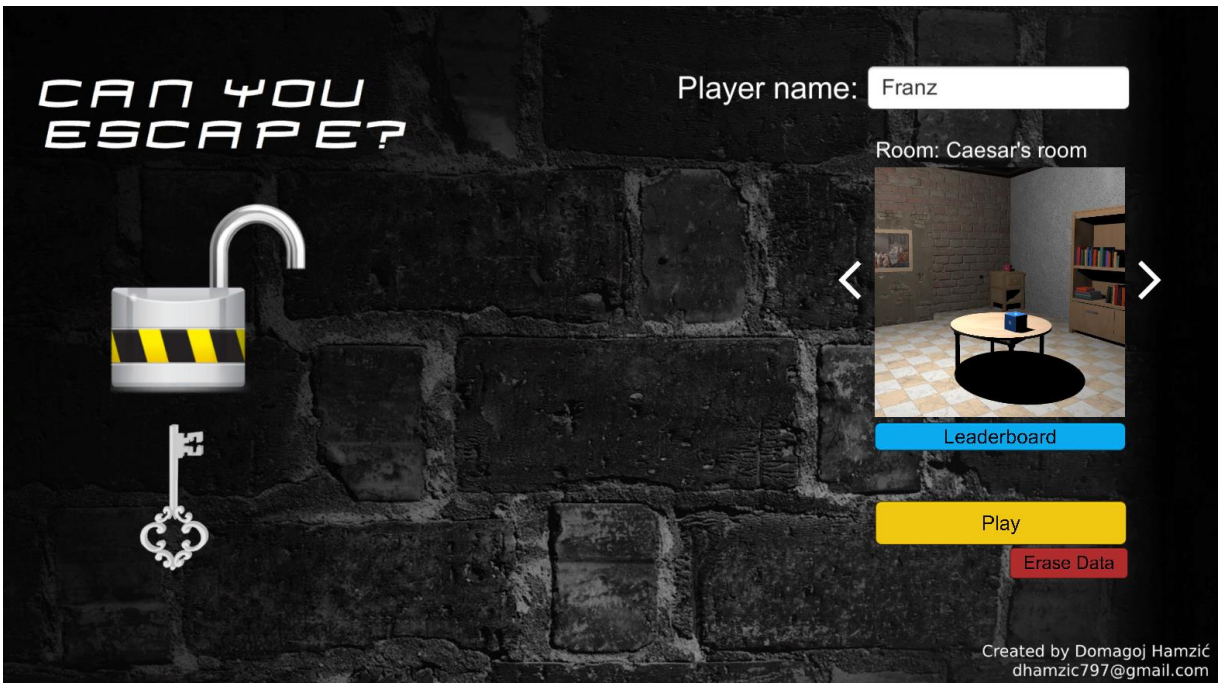
```
25      void Start()
26      {
27          leaderBoardPanel = GameObject.Find("TablePanel");
28          leaderBoardPanel.SetActive(false);
29
30          this.GetRooms();
31
32          if (PlayerPrefs.HasKey("PlayerName"))
33          {
34              this.playerName.text = PlayerPrefs.GetString("PlayerName");
35          }
36      }
```

*Slika 51. LaunchManager.Start [autorski rad]*

Popis soba je hardkodiran. Svaka soba ima naziv i sliku koja će se prikazati korisniku kod odabira iste. Prvo se naprave instance objekta klase Room te se instanciraju nazivi i slike soba. Nakon dodavanja kreiranih soba u listu, korisniku se za početak prikazuje prva soba u listi, točnije soba sa nultim indeksom liste soba.

```
43      public void GetRooms()
44      {
45          Room room1 = new Room()
46          {
47              Name = "Caesar's room",
48              Logo = Resources.Load("Room1_Image", typeof(Sprite)) as Sprite
49          };
50          Room room2 = new Room()
51          {
52              Name = "NoName room",
53              Logo = Resources.Load("Room2_Image", typeof(Sprite)) as Sprite
54          };
55          listOfRooms.Add(room1);
56          listOfRooms.Add(room2);
57
58          roomImage = GameObject.Find("Image").GetComponent<Image>();
59
60          roomImage.sprite = this.listOfRooms[0].Logo;
61
62          roomText = GameObject.Find("UiRoomText").GetComponent<Text>();
63          roomText.text = "Room: " + this.listOfRooms[0].Name;
64      }
```

*Slika 52. Dohvaćanje soba [autroski rad]*




*Slika 53. Glavni meni [autorski rad]*

Odabir soba implementirano je pomoću dvije tipke (Next, Previous). Ukoliko korisnik želi sljedeću sobu, potrebno je povećati trenutni indeks liste (currentListIndex) za 1, a u slučaju da je došao do kraja liste, trenutni indeks liste se resetira na nulti indeks. Ukoliko korisnik želi prethodnu sobu, potrebno je smanjiti trenutni indeks liste za 1, a u slučaju da je došao do početka liste (nulti indeks), trenutni indeks se postavlja na zadnji indeks liste.

```
66      public void NextRoom()
67      {
68          if (this.listOfRooms.Count - 1 == this.currentListIndex)
69          {
70              this.currentListIndex = 0;
71          }
72          else
73          {
74              this.currentListIndex++;
75          }
76          roomImage.sprite = this.listOfRooms[currentListIndex].Logo;
77          roomText.text = "Room: " + this.listOfRooms[currentListIndex].Name;
78      }
        0 references
79      public void PreviousRoom()
80      {
81          if (this.currentListIndex == 0)
82          {
83              this.currentListIndex = this.listOfRooms.Count - 1;
84          }
85          else
86          {
87              this.currentListIndex--;
88          }
89          roomImage.sprite = this.listOfRooms[currentListIndex].Logo;
90          roomText.text = "Room: " + this.listOfRooms[currentListIndex].Name;
91      }
```

*Slika 54. Odabir soba [autroski rad]*

Metoda ShowResults dohvaća pomoću LINQ upita, najboljih deset rezultata odabrane sobe iz PlayerPrefabs klase te puni tablicu rezultata sa istim pomoću foreach petlje. Vremena su u JSON zapisana kao float tip podatka tako da je potrebno prikazati (formatirati) to vrijeme korisniku u što prirodnijem izgledu (132. linija koda).

```
109     private void ShowResults(string roomName)
110     {
111         List<Data> resultsFromSpecficRoom = this.resultList
112             .Where(n => n.RoomName == roomName)
113             .OrderBy(t => t.Time)
114             .Take(10)
115             .ToList();
116
117         Text positionText = GameObject.Find("UiPositionText").GetComponent<Text>();
118         Text timeText = GameObject.Find("UiTimeText").GetComponent<Text>();
119         Text nameText = GameObject.Find("UiNameText").GetComponent<Text>();
120
121         roomText.text = "";
122         positionText.text = "";
123         timeText.text = "";
124         nameText.text = "";
125
126         int count = 1;
127         foreach (Data result in resultsFromSpecficRoom)
128         {
129             TimeSpan time = TimeSpan.FromSeconds(result.Time);
130
131             positionText.text = positionText.text + count + ". " + "\n";
132             timeText.text = timeText.text + time.ToString(@"mm\:ss\:fff") + "\n";
133             nameText.text = nameText.text + result.PlayerName + "\n";
134             count++;
135         }
136     }
```

*Slika 55. Tablica rezultata [autorski rad]*

## 3.2.6.Inventory.cs

Glavni zadatak ove skripte je pohrana objekata u Inventory, dohvaćanje liste objekata te brisanje objekata iz Inventory-a. Globalno je definirana lista objekata koja je tipa Item. Item klasa ima definirana sljedeća svojstva:

```
7   public enum ItemType {
8       SilverKey,
9       GoldenKey,
10      Sunglasses,
11      CassetteTape,
12      SvahiliNote,
13      Book
14  }
15  public ItemType itemType;
16  public string description;
17  public Texture inventoryImage;
18  public int id;
```

*Slika 56. Item klasa [autorski rad]*

Nakon dodavanja novog objekta u Inventory, potrebno je osvježiti prikaz korisniku, a to se postiže pomoću EventHandler-a. U sljedećem naslovu UiInventory.cs biti će detaljnije objašnjeno što se događa kad se okine OnItemListChanged Event.

```
23  public void AddItem(Item item)
24  {
25      itemList.Add(item);
26      OnItemListChanged?.Invoke(this, EventArgs.Empty);
27  }
```

*Slika 57. AddItem [autorski rad]*

Sljedeći izrezak koda dohvaća listu svih objekata koji su trenutno prikupljeni i nalaze se u Inventory-u.

```
29  public List<Item> GetItemList()
30  {
31      return itemList;
32  }
```

*Slika 58. GetiItemList [autorski rad]*

Kao i kod dodavanja objekta u Inventory, nakon brisanja je potrebno pozvati EventHandler kako bi osvježio prikaz Inventory-a korisniku.

```
34  public void RemoveItem(Item item)
35  {
36      itemList.Remove(item);
37      OnItemListChanged?.Invoke(this, EventArgs.Empty);
38  }
```

*Slika 59. RemoveItem [autorski rad]*

### 3.2.7.UiInventory.cs

Dok je Inventory.cs skripta koja sadrži logiku koja igraču nije vidljiva, UiInventory.cs je skripta koja sadrži logiku prikaza svih elemenata koji se nalaze igraču u Inventory-u. Prije samog korištenja Inventory-a, potrebno je kreirati objekt klase Inventory, a to je implementirano u metodi SetInventory. Osim instanciranja Inventory-a, potrebno je dohvatiti itemSlotContainer koji predstavlja okvir gdje se igraču prikazuju objekti iz Inventory-a i itemSlotTemplate koji predstavlja okvir svakog objekta koji se nalazi u Inventory-u (tri objekta == tri klona itemSlotTemplate-a). U skripti Inventory.cs rečeno je da će u ovom naslovu detaljnije biti objašnjeno što se događa kad se okine OnItemListChanged Event. U liniji 32. vidi se da se prilikom okidanja event-a, poziva Inventory_OnItemListChanged metoda koja poziva RefreshInventoryItems metodu.

```
28      public void SetInventory(Inventory inventory)
29      {
30          this.inventory = inventory;
31
32          inventory.OnItemListChanged += Inventory_OnItemListChanged;
33
34          if (itemSlotTemplate == null || itemSlotContainer == null)
35          {
36              //Okvir gdje se nalaze Item-i
37              itemSlotContainer = transform.Find("itemSlotContainer");
38              //Za svaki Item kreira se klon itemSlotTemplate-a
39              itemSlotTemplate = itemSlotContainer.Find("itemSlotTemplate");
40          }
41      }
```

*Slika 60. SetInventory [autorski rad]*

```
43      private void Inventory_OnItemListChanged(object sender, EventArgs e)
44      {
45          RefreshInventoryItems();
46      }
```

*Slika 61. Inventory_OnItemListChanged [autorski rad]*

Slijedi objašnjenje RefreshInventoryItems metode. Metoda je objašnjena direktno u VS editor-u pomoću komentara.

```
public void RefreshInventoryItems()
{
    //Kod svakog dodavanja objekta crta se novi Inventory prikaz
    //tako da je potrebno obrisati stare kako ne bi došlo do duplikata
    foreach (Transform child in itemSlotContainer)
    {
        if (child == itemSlotTemplate) continue;
        Destroy(child.gameObject);
    }
    int x = 0;
    int y = 0;
    //Razmak između dva okvira item-a
    float itemSlotCellSize = 75f;
    foreach (Item item in inventory.GetItemList())
    {
        //Kreiranje itemSlotTemplate okvir za pojedini item.
        //itemSlotTemplate je dijete itemSlotContainera (razlog 2 argumenta)
        RectTransform itemSlotRectTransform = Instantiate(itemSlotTemplate, itemSlotContainer)
            .GetComponent<RectTransform>();
        itemSlotRectTransform.gameObject.SetActive(true);
        //Pozicija okvira pojedinog itema
        itemSlotRectTransform.anchoredPosition = new Vector2(x * itemSlotCellSize, y * itemSlotCellSize);
        //Dohvaća komponentu slike
        Image image = itemSlotRectTransform.Find("image").GetComponent<Image>();
        //Postavlja sliku trenutno dohvaćenog item-a iz Inventory-a
        image.sprite = item.GetSprite();
        //Postavlja broj tipkovnice za korištenje specificnog item-a
        TextMeshProUGUI uiText = itemSlotRectTransform.Find("actionKey").GetComponent<TextMeshProUGUI>();
        uiText.SetText(item.actionKey);
        //smjer postavljanja okvira za sljedeći item
        x++;
        if (x >= 4)
        {
            x = 0;
            y--;
        }
    }
}
```

*Slika 62. RefreshInventoryItems [autorski rad]*

## 3.2.8.WallWithHidenInfo.cs

Skripta koja sadrži logiku prikazivanja i sakrivanja poruke na zidu. Logika je jednostavna, kreirana je tekstura bez poruke i identična tekstura sa porukom te je potrebno mijenjati teksturu objekta ovisno o potrebi. Svojstvo defaultMaterial sadrži teksturu bez poruke dok hiddenMessageMaterial sadrži teksturu sa porukom. Awake metoda postavlja teksturu bez poruke kao defaultMaterial, a GetDefaultMaterial dohvaća tu istu.

```
public class WallWithHidenInfo : MonoBehaviour
{
    private Material defaultMaterial;
    public Material hiddenMessageMaterial;
    Unity Message | 0 references
    private void Awake()
    {
        Material[] materials = GetComponent<Renderer>().materials;
        defaultMaterial = materials[0];
    }
    3 references
    public Material GetDefaultMaterial()
    {
        return this.defaultMaterial;
    }
}
```

*Slika 63. WallWithHidenInfo [autorski rad]*

## 3.2.9.ObjectManipulationText.cs

Skripta služi za prikazivanje opisa korištenja fokusiranog objekta.


*Slika 64. Opis korištenja objekta [autorski rad]*

Kreiran je Canvas sa tekstom koji je pozicioniran u donjem središnjem dijelu ekrana. Klasa ima dvije metode:

- ShowFloatingText
    - Opis komunikacije sa objektom
- ShowWarningText.
    - Upozorenje

FloatingText ovisi o tag-u fokusiranog objekta (može i o nazivu) te ovisno o tag-u ispisuje određeni tekst.

```
20      public void ShowFloatingText(string objectName, string objectTag)
21      {
22          string text = "";
23          switch (objectTag)
24          {
25              case "ObjectSelectable_Painting":
26                  {
27                      text = "Press E To Throw Painting";
28                      break;
29                  }
30              case "ObjectSelectable_Switch":
31                  {
32                      text = "Press E To Turn On/Off Switch";
33                      break;
34                  }
35              case "ObjectSelectable_Inventory":
36                  {
37                      text = "Press E To Pick Up";
38                      break;
39                  }
```

*Slika 65. ShowFloatingText [autorski rad]*

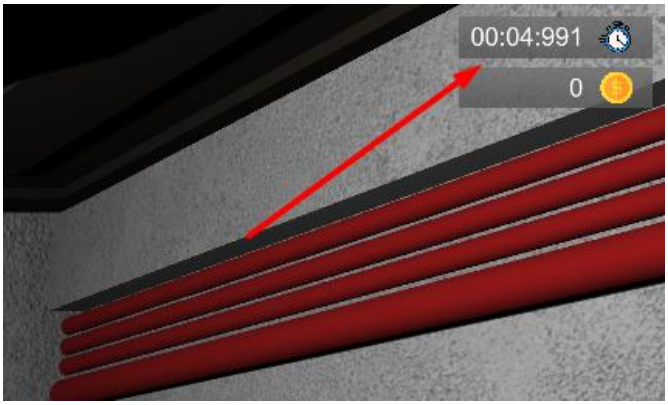
Tekst upozorenja za razliku od teksta opisa, nestaje nakon jedne sekunde prikazivanja, pojavljuje se u gornjem središnjem dijelu ekrana i font je druge boje (crveni). Da bi tekst nestao nakon točno jedne sekunde od prikazivanja, potrebno je koristiti korutine. Korutine su funkcije koje imaju mogućnost pauzirati egzekuciju koda i vratiti kontrolu Unity-u te nastaviti rad nakon što se zadana instrukcija ne završi. Korutine nisu paralelne tj. ne izvršavaju se u posebnoj dretvi (Sve korutine se izvršavaju sekvencijalno u glavnoj dretvi). 83. linija prikazuje željeni tekst upozorenja. 84. linija pauzira egzekuciju koda te nakon toga pokreće postupno nestajanje teksta upozorenja (FadeTextToZeroAlpha). Nakon postepenog nestajanja, tekst upozorenja se resetira na prazan string i postavlja se vrijednost Alphe=1.

```
81      public IEnumerator ShowWarningText(string text)
82      {
83          warningText.text = text;
84          yield return new WaitForSeconds(1f);
85          StartCoroutine(FadeTextToZeroAlpha());
86          yield return new WaitForSeconds(1f);
87          this.warningText.text = "";
88          this.warningText.color = new Color(
89              this.warningText.color.r,
90              this.warningText.color.g,
91              this.warningText.color.b, 1);
92      }
        1 reference
93      public IEnumerator FadeTextToZeroAlpha()
94      {
95          //Alpha==1
96          this.warningText.color = new Color(
97              this.warningText.color.r,
98              this.warningText.color.g,
99              this.warningText.color.b, 1);
100
101         //Smanjuje do 0
102         while (this.warningText.color.a > 0.0f)
103         {
104             this.warningText.color = new Color(
105                 this.warningText.color.r,
106                 this.warningText.color.g,
107                 this.warningText.color.b,
108                 this.warningText.color.a - (Time.deltaTime / 1));
109             yield return null;
110         }
111     }
```

*Slika 66. ShowWarningText [autorski rad]*

### 3.2.10. Stopwatch.cs i Timer.cs

Kao što i sam naziv skripte govori, riječ je o štoperici/brojaču vremena koji su bitni za praćenje rezultata igrača te se pokreće prilikom paljenja scene igre.




*Slika 67. Štoperica [autorski rad]*

Slijedi opis koda direktno u VS editoru.

```
7   public class Stopwatch : MonoBehaviour
8   {
9       bool timerActive = false;
10      public float currentTime;
11      public Text currentTimeText;
12
        Unity Message | 0 references
13      void Start()
14      {
15          //Postavljanje trenutno vremena na 0
16          currentTime = 0;
17          //Pokretanje štoperice
18          StartStopwatch();
19      }
        Unity Message | 0 references
20      void Update()
21      {
22          if (timerActive == true)
23          {
24              //-Time.deltaTime jer 60fps nekad zna biti varijabilan zbog rada CPU-a
25              currentTime = currentTime + Time.deltaTime;
26          }
27          //Pretvorba float tipa podatka u oblik vremena razumljiv igraču
28          TimeSpan time = TimeSpan.FromSeconds(currentTime);
29          currentTimeText.text = time.ToString(@"mm\:ss\:fff");
30      }
31      //Početak igre
        1 reference
32      public void StartStopwatch()
33      {
34          timerActive = true;
35      }
36      //Zaustavljanje štoperice (Gotova Igra)
        1 reference
37      public void StopStopwatch()
38      {
39          timerActive = false;
40      }
41  }
```

*Slika 68. Štoperica kod [autorski rad]*

Timer.cs predstavlja brojač i uspoređujući ju sa Stopwatch.cs, jedina razlika je što Timer.cs u Update metodi, osim zbrajanja vremena (25. linija) oduzima Time.deltaTime od početno definiranog vremena.

```
27              currentTime = currentTime - Time.deltaTime;
28              if (currentTime <= 0) {
29                  StopTimer();
30              }
```

*Slika 69. Timer.cs [autorski rad]*

### 3.2.11. SelectableObject.cs

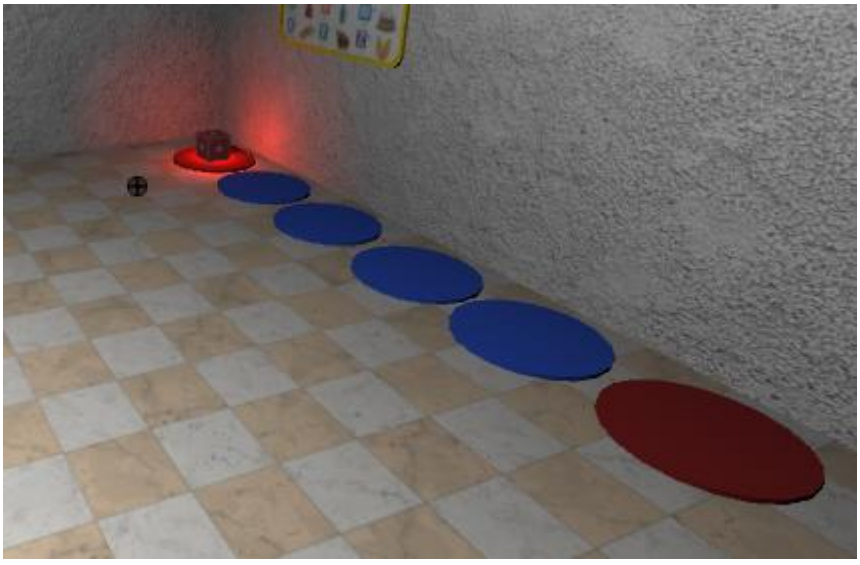
Svaki objekt koji se može selektirati tj. koristiti, mora koristiti skriptu SelectableObject.cs. Primarni zadatak skripte je pohrana materijala/teksture objekta i dohvaćanja iste. Skripta je bitna jer objekt prilikom selektiranja (fokusiranja) poprima material žute boje te je kasnije potrebno vratiti teksturu objekta u prvobitno stanje. Svojstvo locked koristi se kod objekata kojih je potrebno otključati za daljnje korištenje (npr. Ladica, vrata i sl.). Prilikom instanciranja objekta koji koristi ovu skriptu, potrebno je napuniti listu materiala koje taj objekt ima (metoda Awake). Nakon što se makne fokus sa objekta, poziva se metoda GetDefaultMaterials koja vraća prvobitni izgled određenog objekta.

```
10  public class SelectableObject : MonoBehaviour
11  {
12      public bool locked;
13      private List<Material> defaultMaterials = new List<Material>();
            Unity Message | 0 references
14      private void Awake()
15      {
16          Material[] materials = GetComponent<Renderer>().materials;
17          for (int i = 0; i < materials.Length; i++)
18          {
19              defaultMaterials.Add(materials[i]);
20          }
21      }
            2 references
22      public List<Material> GetDefaultMaterials() {
23          return this.defaultMaterials;
24      }
25  }
```

*Slika 70. SelectableObject [autorski rad]*

### 3.2.12. Cylinder.cs

Svaki cilindar koristi ovu skriptu jer primarna funkcija ove skripte je da provjerava koji objekt je stavljen na cilindar. Svaki cilindar očekuje točno specificirani objekt (svojstvo ExpectedCube).



*Slika 71. Cilindri [autorski rad]*

Skripta ima sljedeća svojstva:

- GameObject cylinder – Objekt cilindra koji koristi ovu skriptu
- Char ExpectedCube – Objekt kojeg očekuje taj cilindar, provjerava se zadnji znak
- Bool match – je li postignut pogodak između cilindra i traženog objekta

Metoda OnTriggerStay provjerava koji objekt se sudario sa cilindrom. Ukoliko je riječ o objektu koji ima oznaku „ObjectSelectable_Cube“, algoritam nastavlja sa narednom provjerom, a to je, je li riječ o specifičnom objektu kojeg očekuje specifičan cilindar (Linija 27.). Ako je došlo do pogotka, potrebno je postaviti svojstvo match u true. Prilikom dodira traženog objekta sa cilindrom, pali se svjetlo koje označava da je došlo do kolizije. Metoda OnTriggerExit provjerava micanje objekta sa cilindra. Ukoliko se pomakne specifični objekt sa cilindra, potrebno je ugasiti svjetlo te resetirati svojstvo match u false (Nema pogotka).

```
21      private void OnTriggerStay(Collider other)
22      {
23          if (other.tag == "ObjectSelectable_Cube")
24          {
25              Light sl = cylinder.transform.Find("Point Light").GetComponent<Light>();
26              sl.intensity = 4;
27              if (ExpectedCube == other.name[5])
28              {
29                  this.match = true;
30              }
31          }
32      }
        Unity Message | 0 references
33      void OnTriggerExit(Collider other)
34      {
35          Debug.Log("OnTriggerExit: " + other.name);
36          if (other.tag == "ObjectSelectable_Cube")
37          {
38              Light sl = cylinder.transform.Find("Point Light").GetComponent<Light>();
39              sl.intensity = 0;
40              this.match = false;
41          }
42      }
```

*Slika 72. Cylinder [autorski rad]*

## 3.2.13. Safe.cs

Skripta provjerava unesenu lozinku za sef. Riječ je o statičnoj klasi. Lozinka je hardkodirana (Linija 11). Klasa ima jednu metodu (CheckPassword) koja provjerava je li dobiveni niz znakova (insertedPassword) identičan traženom nizu znakova.

```
9       public static class Safe
10      {
11          public static string Password = "3728";
12
            2 references
13          public static bool CheckPassword(string insertedPassword)
14          {
15              bool correct = false;
16
17              if (insertedPassword == Password)
18              {
19                  correct = true;
20              }
21              return correct;
22          }
23      }
```

*Slika 73. Safe.cs [autorski rad]*

### 3.2.14. Scaler.cs

Skripta je napravljena zbog potrebe korištenja 3D animacije koje imaju različite setove geometrije. Primjerice, uže koje je potrebno spustiti, a animacija spuštanja je riješena pomoću povećanja dužine uža („Extrude“ u Blender-u).

*Slika 74.Prvotna veličina uža [autorski rad]*

*Slika 75. Veličina uža nakon animacije [autorski rad]*

BlendShapes je mehanika Unity-a čiji se ključni okviri animacije kreću u rasponu od 0-100. Svojstvo mSize predstavlja trenutnu vrijednost ključnog okvira (KeyShape). Metoda DropRope pokreće metodu Scale pomoću Unity-eve metode InvokeRepeating koja služi za pokretanje navedene metode u određenim intervalima (u ovom slučaju svake 0.04s). Metoda Scale prilikom svakog okidanja, povećava vrijednost ključnog okvira za 1 sve dok ne dođe do svoje maksimalne vrijednosti, 100 (Animacija gotova, uže je spušteno). Kad je animacija gotova, potrebno je ugasiti pozivanje metode Scale (Linija 19.).

```
5   public class Scaler : MonoBehaviour
6   {
7       private float mSize = 0.0f;
8
        1 reference
9       public void DropRope()
10      {
11          //Svakih 0.04s pomakni za 1 blend shape
12          InvokeRepeating("Scale", 0.0f, 0.04f);
13      }
14
        0 references
15      void Scale()
16      {
17          if (mSize >= 100.0f)
18          {
19              CancelInvoke("Scale");
20          }
21          GetComponent<SkinnedMeshRenderer>().SetBlendShapeWeight(0, mSize++);
22      }
23  }
```

*Slika 76. Scaler [autorski rad]*

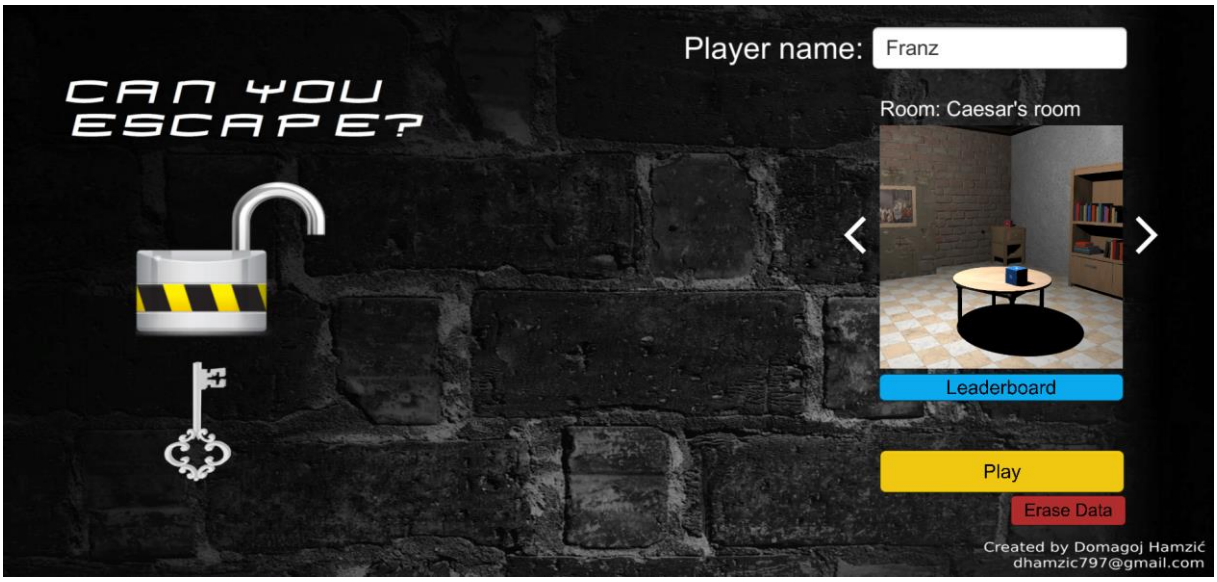
## 3.3. Upute igranja

Slijedi objašnjenje kako pobjeći iz soba koje su kreirane. Za potrebe izrade diplomskog rada, izrađene su dvije sobe:

- Caesar's room
- Morse's room

Korisniku se prvo prikaže glavni meni. Glavni meni sastoji se od prostora za unos imena igrača, odabira sobe i tri tipke:

- Leaderboard
    - Rezultati pojedinačne sobe
- Play
    - Igranje igre, sljedeća scena je igra
- Erase Dana
    - Brisanje svih postignutih rezultata (PlayerPrefs.DeleteAll())




*Slika 77. Main menu [autorski rad]*

### 3.3.1.Caesar's Room

Za razliku od Morse's room, Caesar's room nema ograničeno vrijeme izlaza iz sobe. Ukoliko korisnik želi pauzirati igricu, potrebno je pritisnuti tipku „P“ na tipkovnici. Ukoliko igrač želi koristiti objekte koje je prikupio u Inventory-u, potrebno je pritisnuti tipku „I“. Slijedi objašnjenje savršeno odigrane igre.


*Slika 78. Caesar's room [autorski rad]*

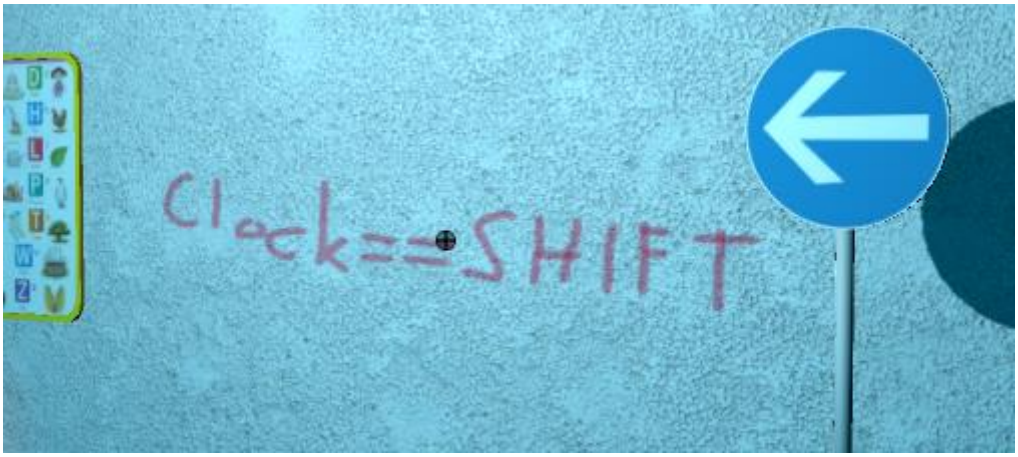
Ispred se nalazi ormar i na njemu ključ kojeg je potrebno uzeti. Ključ otključava zaobljeni ormarić.


*Slika 79. Ormarić [autorski rad]*

U prvoj ladici se nalazi knjiga koja sadrži sljedeći niz znakova: „HVFDSH“ koje trenutno ništa ne govore. U drugoj ladici se nalazi papirić sa oznakom koja je identična kao i slika na prekidaču za svjetlo.

*Slika 80. Hydra [autroski rad]*

U trećoj ladici nalazi se ključ koji se kasnije koristi. Prekidač pali UV lampu koja otkriva skrivenu poruku na istočnom zidu. Isto tako pored poruke se nalazi znak sa obaveznim skretanjem u lijevo. Sat na zidu prikazuje 3h. Riječ je o Caesar shift zagonetki. Svaki znak u nizu znakova „HVFDSH“ potrebno je pomaknuti za 3 polja u lijevo. „H“ postaje „E“, „V“ postaje „S“, „F“ postaje „C“, „D“ postaje „A“, „S“ postaje „P“ i „H“ postaje „E“. Rješenje algoritma je „ESCAPE“. Ukoliko postoje igrači koji imaju problema sa abecedom, lijevo od skrivene poruke „Clock==SHIFT“ nalazi se slika sa engleskom abecedom.




*Slika 81. Caesar Shift [autorski rad]*

U sobi se nalaze kocke sa slovima te se svako slovo nalazi u nizu znakova „ESCAPE“. Potrebno je kocke poredati po cilindrima tako da čine niz znakova „ESCAPE“.

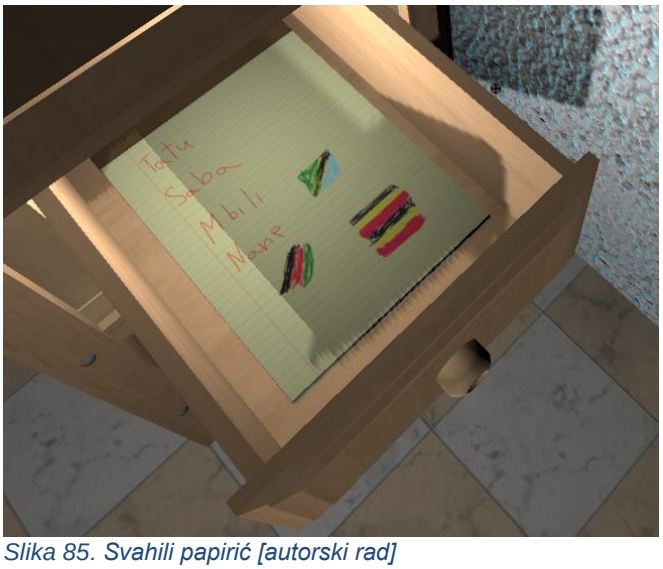
*Slika 82. "ESCAP_" [autorski rad]*

Nedostaje jedna kocka, točnije zadnja sa slovom „E“. Potrebno ju je pronaći. Kocka je skrivena iza slike Cezara. Potrebno je baciti sliku.

*Slika 83. Cezarova slika [autorski rad]*

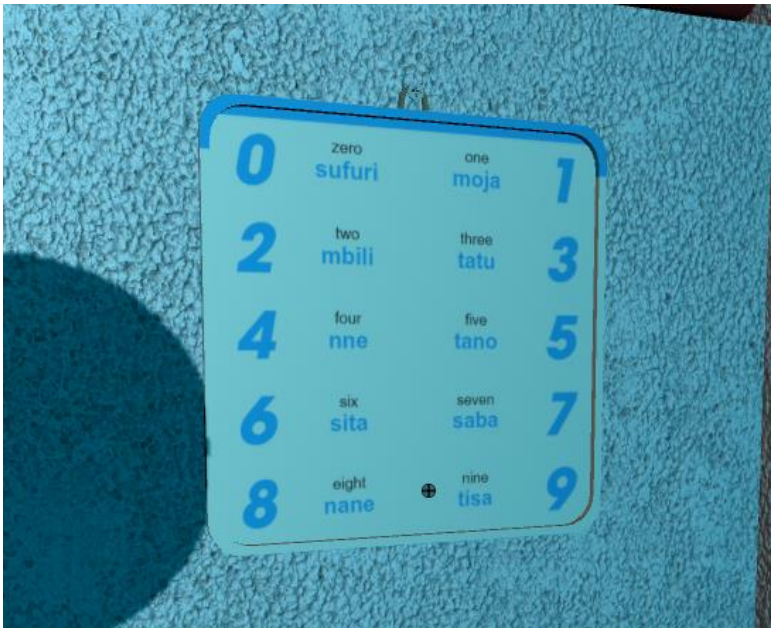
Iza slike se nalazi sef koji traži lozinku od četiri znaka. Desno od sefa nalazi se ormarić koji još uvijek nije istražen. U ladici se nalazi papirić sa nepoznatim znakovima.

*Slika 84. Sef [autorski rad]*




*Slika 85. Svahili papirić [autorski rad]*

U sobi je pronađena slika koja nije istražena. Na slici se nalaze riječi, identične kao i na papiriću te je svaka riječ mapirana sa brojem. „Tatu“ je „3“, „Saba“ je „7“, „Mbili“ je „2“ i „Nane“ je „8“. Šifra za otvaranje sefa je sljedeći niz brojeva: 3728.




*Slika 86. Svahili slika [autorski rad]*

Unutar sefa nalazi se tražena kocka koju je potrebno staviti na prazan cilindar. Nakon postavljanja kocke, ormar sa knjigama se otvara te otkriva skrivena vrata koja je potrebno otključati sa ključem koji je ranije prikupljen iz treće ladice zaobljenog ormarića uz istočni zid. Nakon izlaska iz sobe, igraču se otvaraju informacije o odigranoj igri.

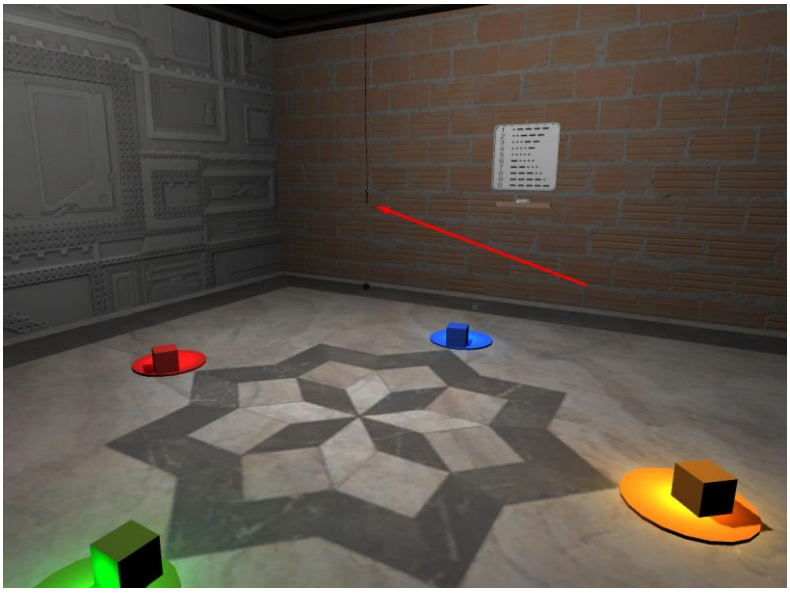

*Slika 87. Rezultat [autorski rad]*

### 3.3.2.Morse's Room

Za razliku od prethodno opisane sobe, izlazak iz ove sobe je vremenski ograničen. Ukoliko igrač ne stigne izaći na vrijeme, igra se zaustavlja tj. prekida.

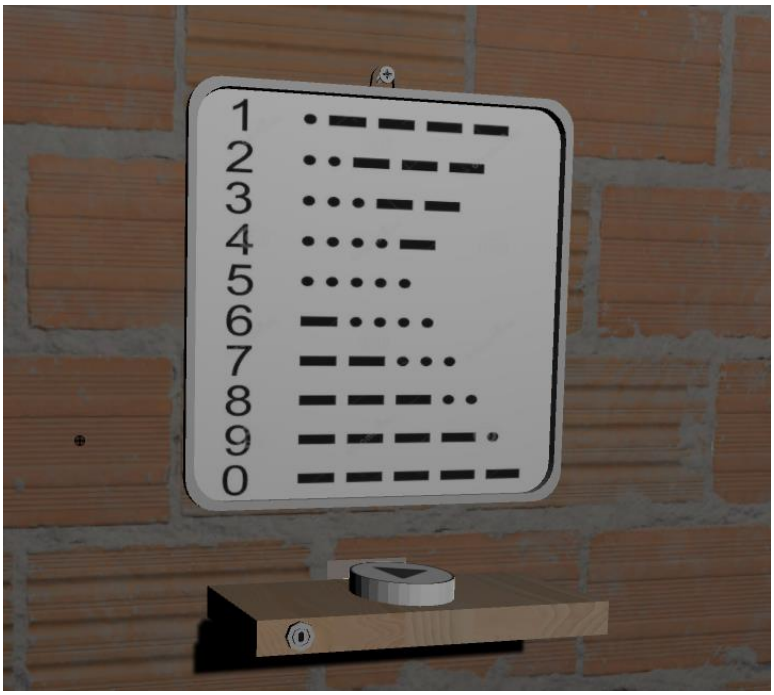

*Slika 88. Morse's Room [autorski rad]*

Potrebno je postaviti svaku kocku na boju cilindra koja je jednaka boji kocke. Kocke su smještene na ladici koja je igraču trenutno nedohvatljiva. Da bi dohvatio kocke, potrebno je stisnuti prekidač koji se nalazi desno od police sa kockama. Prekidač pomiče policu sa koje kao rezultat, ispadaju kocke.  Nakon toga je potrebno postaviti kocke na ispravnu boju.



*Slika 89. Postavljene kocke [autorski rad]*

Nakon što se kocke postave na odgovarajući cilindar, uže sa ključem se spušta te je moguće uzeti ključ i spremiti ga u Inventory. Ključ otključava ladicu koja se nalazi ispod slike Morseovog koda.




*Slika 90. Morseov kod [autorski rad]*

Nakon pritiska tipke (Simbol Play), igraču se pusti zvuk Morseovog koda koji predstavlja lozinku za otključavanje ladice koja se nalazi u desnom gornjem kutu istočnog zida. Potrebno je pomoću slike Morseovog koda i zvuka, otkriti o kojoj je 4-znamenkastoj lozinki riječ. Ispravna lozinka je 9621. Dobivenu lozinku treba ukucati u polje za unos koje se nalazi na ranije navedenoj ladici. Objekt ladice (ne uključujući prostor za unos lozinke) preuzet je sa sljedeće stranice: https://free3d.com/3d-model/drawer-bed-side-547686.html

*Slika 91. Ladica [autorski rad]*

Unutar ladice se nalazi daljinski upravljač koji zahtjeva unos od tri broja. Na zidu se nalaze tri sata od kojih jedan prikazuje 3h, drugi 8h i treći 5h. To su brojevi koje je potrebno unesti na daljinski upravljač i nakon toga pritisnuti tipku „OPEN“.


*Slika 92. Daljinski [autorski rad]*

Nakon unosa tražene kombinacije brojeva i pritiska tipke „OPEN“, iz poda se uzdiže cilindar sa ključem na vrhu. Ključ otključava vrata koja predstavljaju kraj igre.

*Slika 93. Ključ [autorski rad]*

# 4. Zaključak

Prije početka izrade diplomskog rada, bilo je potrebno naučiti osnove rada u Unity-u i proširiti znanje izrade animacija u alatu Blender. Za potrebe učenja, koristio sam dva tečaja:

- Master Procedural Maze & Dungeon Generation [4]
- Build A Multiplayer Kart Racing Game In Unity V.2019 [5]

Teksture objekta preuzete su sa sljedećih stranica:

- https://ambientcg.com/
- https://www.textures.com/library

Proces učenja započeo je 10.04.2021., tako da je proces učenja i izrade diplomskog rada trajao cca. četiri mjeseca. Proces izrade moguće je vidjeti na javnom GitHub repozitoriju koji je napravljen u svrhu izrade diplomskog rada: https://github.com/dhamzic/UnityGame3D

Izvršna datoteka igrice nalazi se na sljedećoj Google Drive poveznici: Escape room

Alat Unity idealan je za početnike jer pruža jednostavno sučelje uz mnogo ugrađenih Unity mehanizama koji olakšavaju izradu poslovne logike igre. Alat Blender je također vrlo jednostavan za izradu modela te je vrlo kompatibilan sa Unity-em tako da je izvoz tj. uvoz modela također riješen vrlo elegantno i bezbolno.

Što se tiče mojeg osobnog mišljenja, trenutno izrađena igra ima mnogo prostora za napredak, možda i koja ideja više što se tiče zagonetki, pogotovo veća mašta prilikom izrade priče igre. Za izradu ozbiljnije igre, ipak je potrebno više ljudi, minimalno troje gdje bi jedna osoba bila zadužena za izradu modela/animacija, druga za poslovnu logiku i treća za kreiranje priče igre.

# Popis literature


[1] D. Hamzić, „Trodimenzionalno modeliranje objekta za stanovanje i uređenje okoliša u Blenderu“, info:eu-repo/semantics/bachelorThesis, University of Zagreb. Faculty of Organization and Informatics. Department of Computing and Technology, 2019. Pristupljeno: kol. 25, 2021. [Na internetu]. Dostupno na: https://urn.nsk.hr/urn:nbn:hr:211:231339

[2] „Tiles 013 on ambientCG“. https://ambientCG.com/ (pristupljeno kol. 25, 2021).

[3] „Cinderblock Closeup - PBR0719“. https://www.textures.com/download/PBR0966/140839 (pristupljeno kol. 26, 2021).

[4] „Master Procedural Maze & Dungeon Generation“, *Udemy*. https://www.udemy.com/course/procedural-maze-dungeon-generation/ (pristupljeno ruj. 01, 2021).

[5] „Build A Multiplayer Kart Racing Game In Unity V.2019“, *Udemy*. https://www.udemy.com/course/kart-racing/ (pristupljeno ruj. 01, 2021).

# Popis slika