IZRADA KARTAŠKE IGRE U UNITY OKRUŽENJU

Domjanović, Josip

SVEUČILIŠTE U SPLITU

SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE

Stručni prijediplomski studij Računarstvo

JOSIP DOMJANOVIĆ

Z A V R Š N I R A D

IZRADA KARTAŠKE IGRE U UNITY OKRUŽENJU

Split, rujan 2024.

SVEUČILIŠTE U SPLITU

SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE

Preddiplomski stručni studij Računarstvo

Predmet: Strukture podataka i algoritmi

Z A V R Š N I R A D

Kandidat: Josip Domjanović

Naslov rada: Izrada kartaške igre u Unity okruženju

Mentor: dr.sc. Toma Rončević prof. struč. stud.

Split, rujan 2024.

Sadržaj

Sažetak 1
1. Uvod 2
2. Pravila igre trešeta 4
3. Razvojno okruženje Unity 7
4. Implementacija igre 9
4.1. 2D Grafičko sučelje 9
4.1.1. Glavni grafički elementi 9
4.1.2. Proces izrade nove karte i ostalih GameObjecta 12
4.1.3. Tijek igre 13
4.2. Računalno upravljani igrači 23
4.2.1. Struktura klase PlayerPrefab 23
4.2.2. Funkcionalnosti klase PlayerPrefab 24
4.2.3. Implementacija Tučem i Strišo 29
5. Zaključak 32
6. Literatura 33

Sažetak

Ovaj završni rad opisuje izradu igre Trešeta u Unity okruženju, s četiri igrača, od kojih su tri računalno upravljana, a jedan korisnički. Cilj rada je razviti cjelovitu implementaciju igre Trešeta, uključujući: 2D grafičko sučelje, računalne igrače s pamćenjem povijesti igre, pravila igre Trešeta, igru u parovima i komunikaciju među igračima i pamćenje bodova preko više partija(jedna partija označava jedno dijeljenje karata i 10 ruku odigranih) do 41 boda.

Rad pruža sveobuhvatan uvid u proces razvoja složene računalne igre u Unity okruženju, s naglaskom na implementaciju ad-hoc pravila za računalne igrače i izradu funkcionalnog i estetski privlačnog korisničkog sučelja. Primjeri i metode opisani u radu mogu se primijeniti u budućim sličnim projektima.

Ključne riječi: 2D grafika, Pamćenje povijesti igre, Računalno upravljani igrači, Trešeta, Unity

Summary

Creating a card game in the Unity environment

This final paper describes the creation of the game Tresseta in a Unity environment, with four players, three of whom are controlled by the computer, and one by the user. The goal was to develop a complete implementation of the Tresseta game, including: 2D graphical interface, computer players with memory of the game's history, complete rules of the Tresseta game, doubles game and communication between players and memory of points over multiple games (one game means one deal of cards and 10 hands played) to a total score of 41 points.

The work provides comprehensive insight into the process of developing a complex computer game in the Unity environment, with an emphasis on implementing ad-hoc rules for computer players, balancing the game, and creating a functional and aesthetically appealing user interface. The examples and methods described in the paper can be applied in future similar projects.

Keywords: 2D graphics, Game history tracking, Computer players, Tressette, Unity

1. Uvod

U ovom završnom radu želi se opisati proces izrade igre Trešeta u Unity okruženju, namijenjene za četiri igrača, od kojih su tri računalna, a jedan je korisnik. Motivacija za odabir ovog projekta proizlazi iz želje za kreiranjem autentičnog iskustva tradicionalne kartaške igre u digitalnom obliku, omogućavajući igračima da se natječu protiv računala koja koriste sofisticirane algoritme za donošenje odluka, također ovaj projekt može poslužiti prilikom učenja igranja ove kartaške igre. Glavni cilj je stvoriti cjelovitu implementaciju igre koja uključuje intuitivno i pregledno 2D grafičko sučelje, računarske igrače s naprednim strategijama temeljenima na povijesti igre, te detaljno implementirana pravila igre Trešeta.

Konkretno, problem je riješen kroz razvoj nekoliko ključnih komponenti:

- **2D grafičko sučelje** - Kreirano je korisničko sučelje koje omogućava jasno praćenje stanja igre u stvarnom vremenu. Koristeći razvojno okruženje Unity, dizajnirano je sučelje koje igračima pruža pregledan prikaz svih važnih elemenata igre, poput trenutnih karata na stolu, bodova i povijesti poteza. Sučelje je dizajnirano tako da bude intuitivno, omogućujući igračima lako razumijevanje igre i donošenje odluka.
- **Računalno upravljani igrači** - Implementirani su algoritmi za računalne igrače koji mogu pamtiti povijest igre i donositi odluke na temelju prethodnih poteza, što simulira realističnu strategiju. Ovi algoritmi koriste ad-hoc pravila kako bi analizirali dosadašnje poteze i predviđali optimalne akcije. Na taj način, računalni igrači se prilagođavaju dinamici igre, što stvara izazovnu i realističnu igru za korisnika.
- **Pravila igre Trešeta** - Sva pravila igre, uključujući specifičnosti poput "akužavanja", detaljno su implementirana kako bi se postiglo autentično iskustvo. Pravila su kodirana tako da pokrivaju sve aspekte igre, od osnovnih poteza do specifičnih situacija i bodovanja. Ova detaljna implementacija osigurava da igra vjerno reproducira tradicionalnu Trešetu, pružajući igračima potpuno uranjanje u igru.

- **Igra u parovima** - Razvijen je sustav koji omogućava igru u parovima, uključujući komunikaciju i koordinaciju strategija među igračima. Ovaj sustav omogućuje da igrači surađuju i planiraju zajedničke strategije. Implementacija komunikacijskih protokola između igrača omogućava realističnu koordinaciju i međusobnu podršku, čineći igru dinamičnijom i društvenijom.
- **Pamćenje bodova** - implementiran je mehanizam za praćenje bodova kroz više partija, sve do konačnog rezultata od 41 boda. Ovaj sustav omogućava kontinuirano praćenje napretka igrača, bilježeći rezultate svake partije i kumulativno zbrajajući bodove. Na taj način, igra pruža dugotrajnije i kompetitivnije iskustvo, omogućujući igračima da prate svoj napredak kroz niz igara.

Glavni doprinos ovog rada je pružanje sveobuhvatnog uvida u proces razvoja složene računalne igre u Unity okruženju. Rad želi detaljno opisati sve korake, uključujući dizajn, programiranje, testiranje i optimizaciju, te integraciju grafike, korisničkog sučelja i umjetne inteligencije. Poseban naglasak stavljen je na implementaciju ad-hoc pravila za računalne igrače, omogućujući stvaranje dinamične i izazovne igre koja simulira ljudsko razmišljanje i strategiju.

Dodatno, rad koristi standardna pravila igre kako bi se osigurale jednake šanse za sve igrače. Posebna pažnja posvećena je izradi funkcionalnog i estetski privlačnog korisničkog sučelja, dizajniranog da bude intuitivno i pregledno.

Metode i pristupi korišteni u ovom radu mogu poslužiti kao temelj za buduće projekte razvoja sličnih igara, kao što je briškula i bela. Opisani procesi i rješenja pružaju praktične smjernice za razvojne programere, uključujući primjere kodova, algoritama i dizajnerskih rješenja, te uvid u kreativni proces i donošenje odluka u razvoju računalnih igara.

Nakon uvod u drugom poglavlju će biti opisana pravila igre trešeta. Treće poglavlje se bavi razvojnim okruženjem Unity. U četvrtom poglavlju se opisuje implementacija grafike i pozadine rada igre. Na kraju u zadnjem poglavlju je dan zaključak rada.

2. Pravila igre trešeta

Trešete, poznata i pod nazivom "trešeta" u Dalmaciji, je tradicionalna talijanska kartaška igra koja je stekla veliku popularnost u Istri, hrvatskom priobalju i otocima. Ime igre potječe od talijanskog izvornika "Tressette" ili "Tresette," što znači "tri sedmice". Na našim prostorima ova igra se najčešće igra s tršćanskim kartama (tal. Carte Triestine), koje sadrže 40 karata u jednom "macu" (špilu). Postoje dvije glavne varijante igre: trešete s akužama (zvanjima ili akužanjem), koje se obično igra do 41 punta, i trešete bez akužanja, koje se igra do 31 punta. Klasična verzija igre igra se u parovima, dva protiv dva, ali se može igrati i u dvoje.

Miješanje i dijeljenje karata u trešeti je ceremonija koja zahtijeva pažnju i preciznost. Svi igrači naizmjenično miješaju i dijele karte, a ispravno je dijeliti karte u smjeru kazaljki na satu.

Trešete se igra s kartama koje su podijeljene u četiri boje ili zoga (u daljnjem tekstu boja): kupe, baštone, špade i dinare. Unutar svake boje postoje karte od broja 1 - 7 i od 11 - 13. U hijerarhiji snage unutar jednog boja, najjača je trica (3), slijede duja (2), as (1), kralj (13), konj (12), fanat (11), a zatim karte od sedmice (7) prema četvorki (4).

Trešeta kao igra ima jednostavan sistem bodovanja. Karte u rasporedu od četvorke do sedmice nemaju nikakvu vrijednost i one se zovu „lišine“. Karte koje su u rasporedu od jedanaest do trinaest i dvojka te trojka imaju vrijednost od jedne „bele“, odnosno trećine boda. Najveću vrijednost u cijeloj igri ima karta As, odnosno jedinica, koja ima vrijednost jednog boda. U igri postoje i dodatni bodovi koji su omogućeni kroz „akužavanje“ i zadnji bod.

Igra se igra dok jedan od timova ne skupi 41 bod. Kad jedan tim postigne ovaj cilj, bodovi se poništavaju i igra kreće iz početka. Ovaj ciklus omogućuje kontinuiranu igru i stalnu priliku za oba tima da pokažu svoju strategiju i vještine. Kroz takav sustav bodovanja osigurava se dinamična i izazovna igra koja zahtijeva pažljivu koordinaciju i planiranje od strane oba tima. U igri s četiri igrača, svaki igrač dobiva deset karata. Prije nego što odigraju kartu, igrači mogu zvati „akužu“. Igra se tako da igrači naizmjenično bacaju po jednu kartu,

a zatim najjača strana pokupi bačene karte. Tako završava jedna "ruka" te partije, a sljedeću ruku započinje igrač koji je uzeo prethodnu ruku najjačom kartom.

Nakon što se sve karte za igrača podijele, odmah se provjerava ima li igrač uvjete za „akužu“. „Akužavanje“ predstavlja zvanje kombinacija karata koje donose dodatne punte. As, trica i duja iste boje čine „napolitanu“ i nose tri punta. Tri ili četiri iste karte (asa, duje ili trice) također donose punte. Akuža se obično objavi prije nego što igrač baci prvu kartu u prvoj ruci, iako postoje varijante gdje je dozvoljeno akužavanje tijekom cijele igre.

Trešeta je kartaška igra bogata taktikom i strategijom, koja zahtijeva od igrača ne samo vještinu u igranju karata, već i sposobnost preciznog praćenja bodova i pravilnog planiranja poteza. Kako bismo osigurali da igra teče glatko i po pravilima, bitno je razumjeti sve ključne faze igre, od miješanja i dijeljenja karata, preko bacanja i skupljanja karata, do računanja konačnih bodova. Pamćenje bodova igra ključnu ulogu u određivanju pobjednika svake partije. Sustav bodovanja temelji se na preciznim pravilima koja definiraju vrijednost svake karte i kombinacije karata, kao i dodatne bodove koje igrači mogu osvojiti tijekom igre odnosno „akužavanja“. Uvođenjem sustava za pamćenje bodova unutar aplikacije, omogućeno je automatsko i točno bilježenje bodova, što smanjuje mogućnost pogrešaka i omogućava igračima da se fokusiraju na samu igru. [1]

U igri trešeta, komunikacija između igrača je ograničena na nekoliko unaprijed dogovorenih signala kako bi se održala fer igra i strategija. Dva najvažnija signala u ovoj igri su "tučem" i "strišo," koji igraju ključnu ulogu u koordinaciji između partnera u timu.

Tučem je signal koji znači "ubij najjačom kartom (te boje, očito) i vrati istu boju." Koristi se kada prvi igrač ima dvije od tri najjače karte u određenoj boji (as, duja, trica) i na taj način pita suigrača je li onaj treći adut kod njega. Na primjer, ako igrač ima asa i duju u boji kupa, a nema tricu, može koristiti signal tučem da provjeri ima li njegov partner tricu u toj boji. Ovaj signal također se koristi kada igrač ima samo jednu jaku kartu u nekoj boji, ali može razviti igru ako mu padnu druge jake karte. Nadalje, tučem se koristi pri kraju partije kada igrač zna u kojoj je boji jak njegov partner, ali ne može doći u tu boju. U tom slučaju, traži od partnera da baci punte (bodove) kako ne bi propali.

Strišo je signal s višestrukim značenjima, ovisno o kontekstu igre. U laičkoj varijanti, strišo znači "koju boju hoćeš da ti vratim? tj. u kojoj boji si jak?" Međutim, ozbiljni igrači rijetko koriste strišo u ovom kontekstu jer se time gubi jedna sigurna karta, poznata kao "franjka." U ozbiljnijoj igri, strišo može značiti "doći ću ti," što znači da igrač koji baca kartu signalizira partneru da zna gdje je jak i da će mu, nakon što izigra svoje karte, doći u njegovu boju. Također, strišo može značiti da igrač ne želi da njegov partner uzme njegovu kartu. U tom slučaju, partner treba baciti što manju kartu kao znak da ima najjaču kartu u toj boji i da želi da se ta boja opet odigra.

Primjer upotrebe tučem signala u igri može biti situacija kada prvi igrač baci asa i signalizira tučem, a njegov partner, prepoznajući signal, baca tricu iz iste boje, omogućujući prvom igraču da uzme rundu. S druge strane, strišo može biti korišten kada igrač baci srednju kartu boje u kojem zna da njegov partner ima najjaču kartu. Partner tada može baciti manju kartu, signalizirajući da će se vratiti u tu boju, ili jaču kartu, preuzimajući kontrolu nad rundom.

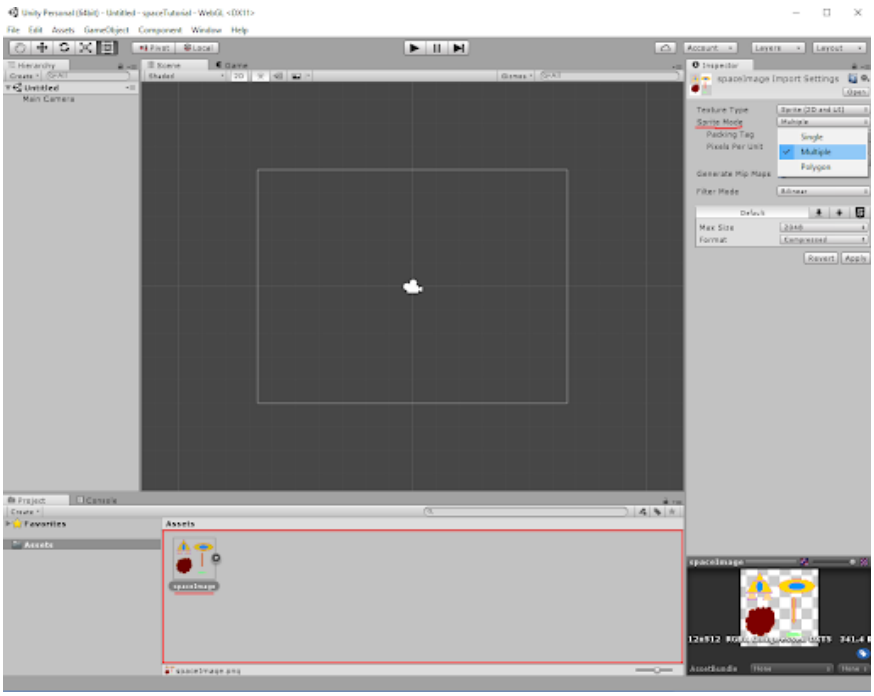
Ovi signali su suptilni, ali bitni elementi igre trešeta koji omogućavaju igračima da razvijaju strategiju i surađuju s partnerima bez verbalne komunikacije. Razumijevanje i pravilna upotreba signala tučem i strišo može značajno povećati šanse za pobjedu i dodaje sloj dubine i taktike ovoj tradicionalnoj kartaškoj igri.

3. Razvojno okruženje Unity

Unity je višeplatformski programski alat za igre koji je razvio Unity Technologies. Prvi put je najavljen i objavljen u lipnju 2005. na Appleovoj svjetskoj konferenciji za razvojne programere kao pogon za igre za Mac OS X. Od tada, Unity je proširen kako bi podržao razne platforme, uključujući stolna računala, mobilne uređaje, konzole, proširenu stvarnost i virtualnu stvarnost. Posebno je popularan za razvoj mobilnih igara za iOS i Android, smatra se jednostavnim za korištenje, te je popularan među nezavisnim programerima igara.

Unity se koristi za stvaranje 3D i 2D igara, kao i interaktivnih simulacija. Osim u video igrama, programski alat je usvojen i u drugim industrijama poput filma, automobilske industrije, arhitekture, inženjeringa, građevinarstva i vojske.

Na slici 1 prikazana je radna površina sučelja za razvoj 2D igre.

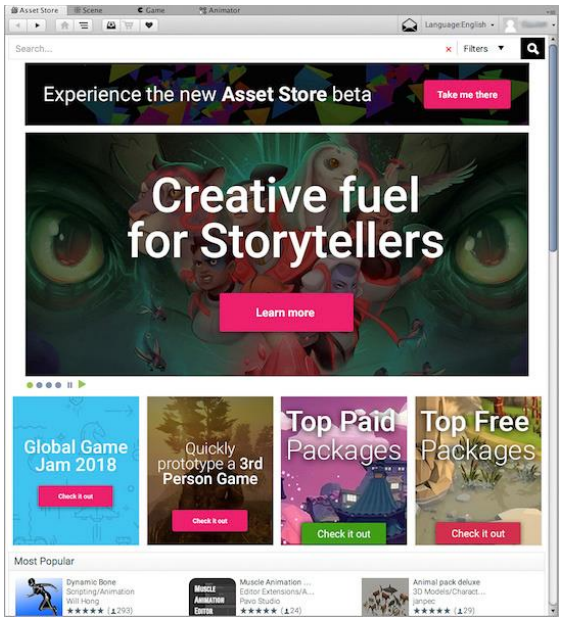


Slika 1: Primjer sučelja za razvoj igre [2]

Unity je postao poznat po svojoj pristupačnosti i svestranosti. Omogućuje korisnicima stvaranje igara za različite platforme s primarnim API-jem za skriptiranje u C#.

Ranije verzije podržavale su i druge jezike poput Boo i UnityScript, ali su te podrške ukinute u korist C#. Unity također nudi napredne alate za renderiranje i optimizaciju za 2D i 3D igre.

Unity Asset Store omogućuje kreatorima da kupuju i prodaju materijale koje su generirali korisnici. Trgovina nudi razne 3D i 2D resurse koji mogu ubrzati razvojne procese. Asset store ima jednostavnu strukturu kako je prikazano na slici 2.



Slika 2: Primjer sučelja Asset Store-a [3]

Unity podržava razvoj na raznim platformama uključujući Windows, macOS, Linux, iOS, Android, te konzole kao što su PlayStation, Xbox i Nintendo Switch. Također podržava VR i AR platforme. Unity koristi različite modele licenciranja, uključujući besplatnu licencu za male tvrtke i pretplatničke modele za veće timove i profesionalne projekte. Besplatna verzija je dostupna za osobnu upotrebu ili tvrtke s manje od 200.000 dolara godišnjeg prihoda. Unity Pro i druge plaćene opcije uključuju dodatne značajke i alate usmjerene na profesionalne projekte. Nedavne promjene u licenciranju, posebno uvođenje "naknade za rad", izazvale su kontroverze među programerima. Naknada bi se naplaćivala po instalaciji igre, što je izazvalo kritike zbog potencijalnog utjecaja na programere, posebno one koji izdaju besplatne igre ili sudjeluju u dobrotvornim akcijama. Unity je odgovorio na povratne informacije programera najavivši izmjene uvjeta kako bi se riješili problemi i osigurala pravednija primjena novih pravila.[4]

4. Implementacija igre

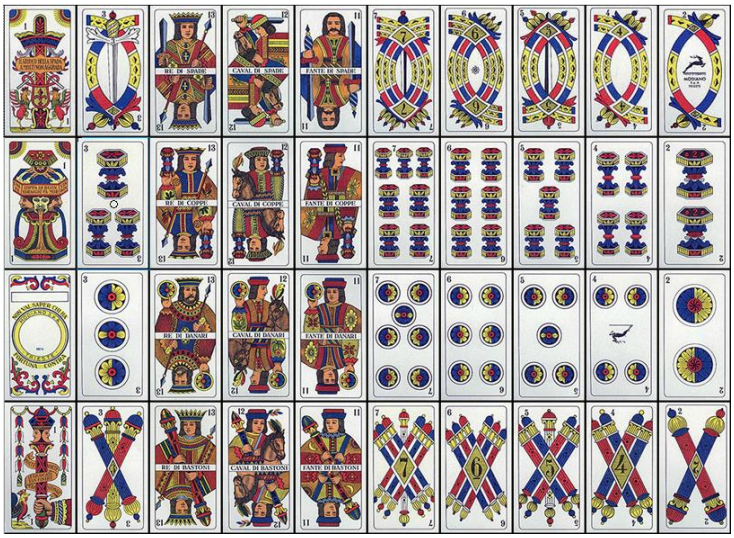
Poglavlje implementacija igre se bavi samom srži ove igre , tj. grafičkim sučeljem iz Unity okruženja i programskim kodom koji se vrti u pozadini. Podrobno su opisane glavne programske značajke i funkcionalnosti s primjerima programskog kôda.

4.1. 2D Grafičko sučelje

Grafičko sučelje je jedan od bitnih elemenata koji čine ovu igru funkcionalnom. Ono omogućuje da korisnik vidi trenutno stanje na stolu te uz pomoć njega može pratiti igru.

4.1.1. Glavni grafički elementi

Za razvoj grafičkih elemenata karata za igru Trešeta, pribavljena je jedinstvena velika slika koja sadrži sve karte potrebne za igru kako je prikazano na slici broj 3. Proces izrade pojedinačnih slika karata uključivao je nekoliko ključnih koraka. Prvo je nabavljena visokokvalitetna slika s kartama u odgovarajućem rasporedu. Zatim je korišten grafički softver za precizno rezanje velike slike na pojedinačne karte. Svaka karta je izrezana i spremljena kao zasebna slika s odgovarajućim nazivom, kako bi bila lako prepoznatljiva i spremna za korištenje u igri. Na kraju, svaka izrezana karta optimizirana je za upotrebu unutar Unity okruženja, što je uključivalo smanjenje veličine datoteke bez gubitka kvalitete, kako bi se osigurala brza učitavanja i glatko izvođenje igre.



Slika 3: Karte korištene za izradu igre[5]

Unity je snažna i svestrana platforma za razvoj igara, koja pruža bogat skup alata za upravljanje grafičkim elementima. U razvoju igre Trešeta korišteno je nekoliko ključnih grafičkih elemenata Unity platforme, uključujući dugmad, canvase i transformacije.

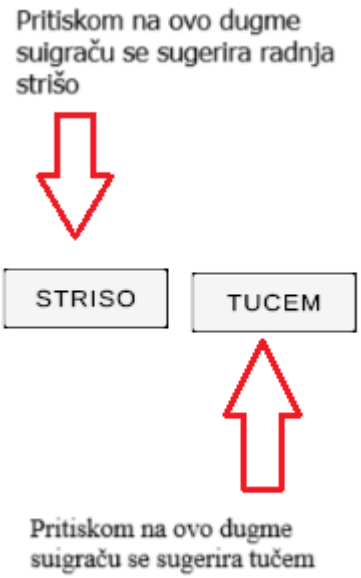
Unity omogućuje jednostavnu izradu i prilagodbu dugmadi. Svako dugme može se personalizirati kroz različite opcije za promjenu boje, oblika, veličine i teksta. Interakcija s dugmadi se također lako podešava putem skriptiranja, omogućujući različite akcije poput pokretanja igre, pauziranja ili otvaranja izbornika. Na primjer, dugmad za početak igre, povlačenje poteza ili pregled pravila igre su dizajnirana da budu intuitivna i lako dostupna igračima, čime se poboljšava cjelokupno korisničko iskustvo.

Posebno, dugmad igra ključnu ulogu u odabiru karata za igrača. Pritiskom na određeno dugme, igrač može odabrati kartu koju želi igrati, a uz pomoć transformacija, ta se karta zatim prebacuje na stol gdje se odvija igra. Transformacije omogućuju precizno pozicioniranje karata na stolu, animaciju njihovog pomicanja i pravilno prikazivanje u igri. Ovaj sustav osigurava da su sve interakcije glatke i vizualno privlačne, čime se dodatno poboljšava korisničko iskustvo i omogućuje dinamičnije i zabavnije igranje.

Unity omogućuje jednostavnu izradu i prilagodbu dugmadi. Svako dugme može se personalizirati kroz različite opcije za promjenu boje, oblika, veličine i teksta. Interakcija s dugmadi se također lako podešava putem skripti, omogućujući različite akcije poput pokretanja igre, pauziranja ili otvaranja izbornika. Na primjer, dugmad za početak igre, povlačenje poteza ili pregled pravila igre su dizajnirana da budu intuitivna i lako dostupna igračima, čime se poboljšava cjelokupno korisničko iskustvo.[6]

Primjer jedne takve interakcije je pokretanje igre pritiskom na dugme "DEAL CARDS". Ova funkcionalnost omogućuje igraču da započne igru jednostavnim pritiskom na dugme, nakon čega se karte automatski dijele i pozicioniraju na stol uz pomoć transformacija.

Također se dugmad koriste za implementaciju komunikacije između korisnika i suigrača, odnosno „AIIgrača2“ Ovakav primjer komunikacije je implementiran uz pomoć dugmadi koja su postavljena na radnoj podlozi. Ovi načini komunikacije se mogu samo koristiti za korisnika i on može samo sugerirati svom suigraču, odnosno samo AIIgraču 2 da koristi radnje „Strišo“ i „Tučem“ uz pomoć dugmadi sa Slike 4.



Slika 4: Slika pozicioniranja i načina korištenja dugmadi.

Canvas je osnovni element za izradu korisničkog sučelja u Unityju. Canvas predstavlja područje unutar kojeg se svi UI elementi, uključujući dugmad, tekst i slike, smještaju i prikazuju. Canvas može biti prilagođen u smislu veličine i pozicioniranja, a Unity pruža opcije za prilagođavanje canvas skaliranja i prilagodbe različitim rezolucijama ekrana. To osigurava da korisničko sučelje ostane dosljedno i funkcionalno na različitim uređajima. U igri Trešeta, canvas je korišten za postavljanje elemenata kao što su igraće karte, rezultati, tajmeri i druge interaktivne komponente koje igrači trebaju vidjeti tijekom igre.[7]

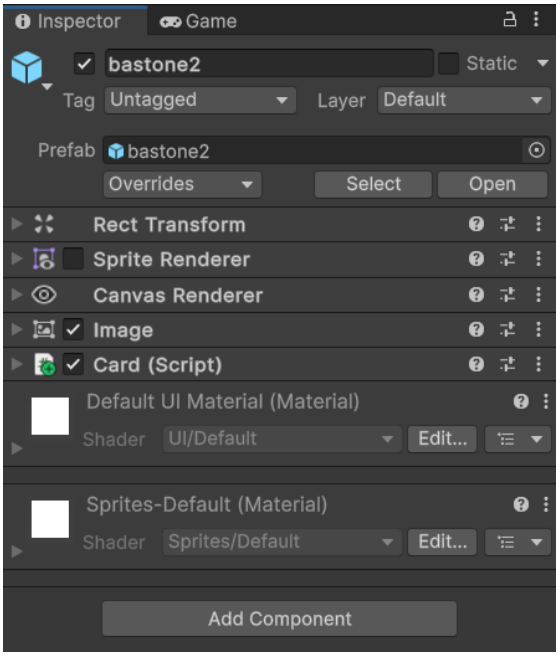
Pored ovih elemenata, Unity nudi i dodatne alate kao što su Particle System za specijalne efekte, Shader Graph za prilagodbu vizualnog izgleda objekata i Animator za složene animacije likova i objekata. Korištenjem ovih alata, grafički elementi u igri Trešeta mogu biti dodatno obogaćeni, pružajući igračima impresivno vizualno iskustvo. Također, Unity Asset Store omogućuje preuzimanje različitih gotovih grafičkih resursa koji mogu ubrzati proces razvoja i dodatno unaprijediti vizualni aspekt igre. Prilikom izrade ove igre Asset store nije mogao biti korišten zbog nedostatka potrebnog sadržaja. [8]

4.1.2. Proces izrade nove karte i ostalih GameObjecta

Prvi korak u ovom procesu je priprema odgovarajućih grafičkih materijala. Priprema uključuje nabavku visokokvalitetne slike karte koja mora biti optimalne rezolucije kako bi se osigurala jasnost i kvaliteta prikaza. Nakon toga, slika se precizno izrezuje na pojedinačne slike karata koristeći grafički softver poput Adobe Photoshopa ili GIMP-a.

Nakon pripreme grafičkog materijala, slike se importiraju u Unity projekt. Da bi se to postiglo, potrebno je otvoriti Unity projekt i unutar Project panela odabrati opciju za importiranje novih „asseta“. Slike karata se zatim odabiru i importiraju u odgovarajuću mapu unutar projekta. Nakon importiranja, slike karata su spremne za korištenje u igri.[9]

Sljedeći korak je kreiranje novog GameObjecta za kartu unutar Unity scene. U Hierarchy panelu se desnim klikom otvara kontekstualni izbornik i odabire se opcija za kreiranje novog GameObjecta. GameObject se zatim imenuje prema karti koju će predstavljati, npr. "As Spadi". Nakon toga, na novi GameObject se dodaje Image komponenta putem Inspector panela koji je prikazana na slici broj 5. Ova komponenta omogućava prikaz slike karte na grafičkom sučelju.

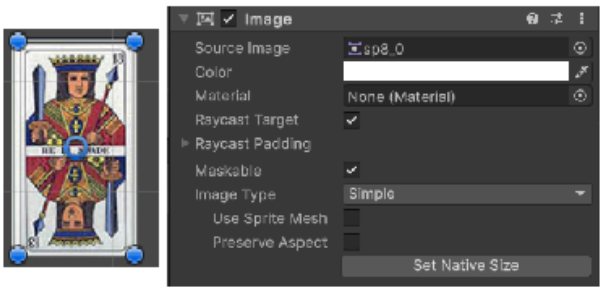


Slika 5 Primjer Unity Inspectora za jednu kartu [10]

Nakon dodavanja Image komponente, potrebno je dodijeliti odgovarajuću sliku karti. Unutar Inspector panela, polje "Source Image" se postavlja na sliku karte koja je prethodno importirana. Dodatno, može se prilagoditi veličina i pozicija slike unutar Scene panela kako bi se osiguralo da karta bude pravilno prikazana unutar korisničkog sučelja.[11]

Kada je karta postavljena i vidljiva na sučelju, slijedi integracija interaktivnosti. Kroz skriptiranje u C#, karta se može učiniti interaktivnom dodavanjem odgovarajućih skripti koje omogućavaju reakciju na korisničke akcije poput klika. Ove skripte mogu uključivati funkcije za odabir karte, pomicanje karte na stol ili aktiviranje animacija.

U konačnici, svaki dodani GameObject koji predstavlja kartu mora biti optimiziran za glatko izvođenje igre. To uključuje smanjenje veličine slike bez gubitka kvalitete, pravilno postavljanje svih potrebnih komponenti slike na GameObject kako bi slika bila ispravno vidljiva. Primjer karte i njezine glavne komponente unutar inspectora je prikazana na slici 6.



Slika 6: Karta i image komponenta unutar inspectora.

Ovaj proces dodavanja nove karte na grafičko sučelje osigurava da igrači imaju vizualno privlačno i funkcionalno korisničko sučelje koje omogućava jednostavnu i intuitivnu interakciju s kartama tijekom igre.

4.1.3. Tijek igre

U ovom poglavlju pružit ćemo iscrpan prikaz kako se odvija sama igra, korak po korak, te kako se bodovi bilježe, računaju i prate tijekom cijele partije.

Tijek igre je zamišljen kroz klasu `GameMenager` koja se standardno koristi unutar Unity okruženja. Unutar te klase se koriste sve prethodno kreirane klase koje su korištene i one se pozivaju u trenutcima kada je to potrebno kako bi se osigurao pravilan raspored igre. Unutar ove klase su dodane i vremenske pauze od nekoliko sekundi, ovisno o potrebi, kako bi se osiguralo da korisnik može pratiti tijek i dinamičnost igre. Također, unutar ove klase se pozivaju i svi „canvasi“, odnosno svi grafički objekti koji se koriste unutar igre. To obuhvaća sva dugmad, sva područja na koja se postavljaju karte i stol na kojem se igra obavlja. Igra počinje pritiskom na dugme „Deal Cards“ odnosno dijeli karte. Na to dugme je postavljen „listener“, odnosno okidač koji povezuje dugme s funkcijom koja se aktivira pritiskom. Ta funkcija pokreće funkciju koja se zove `DealCards` i koja ja zadužena za raspodjelu karata unutar igre i ta funkcija svakom igraču dodjeljuje deset karata. Miješanje karata je primijenjeno funkcijom `ShuffleDeck` koja miješa krate slučajnim odabirom , kako je prikazano na Ispisu 1. Mijenja se pozicija slučajne karte čime se mijenja raspored špila.

```
public void ShuffleDeck()
    {
        for (int i = cards.Count - 1; i > 0; i--)
        {
            int randomIndex = Random.Range(0, i + 1);
            GameObject temp = cards[i];
            cards[i] = cards[randomIndex];
            cards[randomIndex] = temp;
        }

        Debug.Log("Deck shuffled.");
    }
```

Ispis 1: Miješanje karata

Prve karte dobiva korisnika a nakon njega računalni igrači. Ova funkcija se također brine da se svaka karta pravilno dodijeli u ruku svakom igraču kako bi mogao on poslije pravilno je baciti i koristi za analizu igre. Funkcija za podjelu karata je prikazana na Ispisu 2. Nakon što se karte podjele program provjerava imaju li igrači uvjete za „akužavanje“

```
private void DealCards()
    {
        int numCardsPerPlayer = 10;
        float cardWidth = 100f;
        float spacing = 10f;
        float zOffset = -1f;
        currentTurn = 0;
        ResetGameState();
        message.text = "";
        desk.ClearDesk();
        if (deck.cards.Count != numCardsPerPlayer * players.Count)
        {
            Debug.LogWarning("Not enough cards in the deck to deal.");
            return;
        }

        foreach (PlayerPrefab player in players)
        {
            player.hand.Clear();
        }
        deck.ShuffleDeck();

        int cardIndex = 0;
        for (int j = 0; j < 4; j++)
        {
            PlayerPrefab player = players[j];
            GameObject playerGO = gameObjects[j];
            RectTransform playerArea = playerGO.GetComponent<RectTransform>();
            bool isHorizontal = (playerGO == gameObjects[0] || playerGO == gameObjects[2]);

            float startX = playerArea.rect.xMin + cardWidth / 2f + spacing;
            float startY = playerArea.rect.yMin + cardWidth / 2f + spacing;

            for (int i = 0; i < 10; i++)
            {
                if (cardIndex >= deck.cards.Count)
                {
                    Debug.LogWarning("Not enough cards in the deck to deal.");
                    return;
                }
                GameObject drawnCardGO = deck.cards[cardIndex];
                cardIndex++;

                player.AddCardToHand(drawnCardGO);
                drawnCardGO.transform.SetParent(playerGO.transform, false);

                float xPos = startX + i * (cardWidth + spacing);
                float yPos = startY + i * (cardWidth + spacing);

                if (isHorizontal)
                {
                    drawnCardGO.transform.localPosition = new Vector3(xPos, playerArea.rect.yMax - cardWidth / 2f - spacing, zOffset);
                    drawnCardGO.transform.rotation = Quaternion.identity;
                }
                else
                {
                    drawnCardGO.transform.localPosition = new Vector3(playerArea.rect.xMax - cardWidth / 2f - spacing, yPos, zOffset);
                    drawnCardGO.transform.rotation = Quaternion.Euler(0, 0, 90);
                }

                if (player.playerName == "Player1")
                {
                    AddButtonToCard(drawnCardGO);
                }
                else
                {
                    PlayerPrefab.DisableButtonOnCard(drawnCardGO);
                }
            }
```

Ispis 2: Funkcija za podjelu karata

„Akužavanje“ je implementirano unutar klase `PlayerPrefab` s funkcijom koja se zove `HasRequiedCards` , odnosno ima li igrač sve potrebne karte. Funkcija iterira kroz cijelu ruku igrača i provjerava koliko ima trica, dvica (duja) i asova te se sve sprema u tri rječnika za svaku vrijednost. Rječnici sadrže vrijednosti za svaku boju i broj. Također postoji brojač za svaku vrijednost. Implementacija „akužavanja“ je prikazana na Ispisu 3.

```
public int HasRequiredCards(TMP_Text message)
    {
        int result = 0;

        Dictionary<Suit, int> dvicaCountBySuit = new Dictionary<Suit, int>();
        Dictionary<Suit, int> tricaCountBySuit = new Dictionary<Suit, int>();
        Dictionary<Suit, int> asCountBySuit = new Dictionary<Suit, int>();

        int totalDvicaCount = 0;
        int totalTricaCount = 0;
        int totalAsCount = 0;

        foreach (GameObject cardGO in hand)
        {
            Card card = cardGO.GetComponent<Card>();

            if (card.cardValue == Value.DVICA)
            {
                if (!dvicaCountBySuit.ContainsKey(card.cardSuit))
                    dvicaCountBySuit[card.cardSuit] = 0;
                dvicaCountBySuit[card.cardSuit]++;
                totalDvicaCount++;
            }
            else if (card.cardValue == Value.TRICA)
            {
                if (!tricaCountBySuit.ContainsKey(card.cardSuit))
                    tricaCountBySuit[card.cardSuit] = 0;
                tricaCountBySuit[card.cardSuit]++;
                totalTricaCount++;
            }
            else if (card.cardValue == Value.AS)
            {
                if (!asCountBySuit.ContainsKey(card.cardSuit))
                    asCountBySuit[card.cardSuit] = 0;
                asCountBySuit[card.cardSuit]++;
                totalAsCount++;
            }
        }
```

Ispis 3: Provjera potrebnih karata

Nakon što se sve karte prebroje onda se iterira kroz svaku boju (zog) postoji li za nju u ruci igrača trica ,dvica i as. Ako postoji timu tog igrača se dodjeljuje 3 punta, odnosno 9 bela. Nakon toga se provjerava ima li igrač u ruci tri ili četiri karte s vrijednosti as, dvica ili trica, neovisno o boji. Ako ima dodaju se punti njegovom timu. Ako ima tri karte timu se dodjeljuju tri punta, a ako ima četiri karte timu se dodjeljuju četiri punta. Funkcija se nalazi na Ispisu 4.

```
foreach(Suit suit in System.Enum.GetValues(typeof(Suit)))
        {
            if (dvicaCountBySuit.ContainsKey(suit) && tricaCountBySuit.ContainsKey(suit) && asCountBySuit.ContainsKey(suit) &&
                dvicaCountBySuit[suit] >= 1 && tricaCountBySuit[suit] >= 1 && asCountBySuit[suit] >= 1)
            {
                message.text +="\n"+ playerName + " ima akuzu od 3  "+ suit+"  ";
                result += 9;
            }
        }
        if (totalDvicaCount >= 4)
        {
            message.text += "\n" + playerName + " ima akuzu od 4 Dvice  ";
            result += 12;
        }
        else if (totalDvicaCount >= 3)
        {
            message.text += "\n" + playerName + " ima akuzu od 3 Dvice  ";
            result += 9;
        }

        if (totalTricaCount >= 4)
        {
            message.text += "\n" + playerName + " ima akuzu od 4 Trice  ";
            result += 12;
        }
        else if (totalTricaCount >= 3)
        {
            message.text += "\n" + playerName + " ima akuzu od 3 Trice  ";
            result += 9;
        }

        if (totalAsCount >= 4)
        {
            message.text += "\n" + playerName + " ima akuzu od 4 Asa  ";
            result += 12;
        }
        else if (totalAsCount >= 3)
        {
            message.text += "\n" + playerName + " ima akuzu od 3 Asa ";
            result += 9;
        }

        return result;
    }
```

Ispis 4: Brojanje akuži

Na kraju funkcije se vraća rezultat ukupnih bodova koje je taj igrač skupio s svojim kartama. Postoji mogućnost više „akuža“, odnosno može se dogoditi npr. da igrač ima tri karte vrijednosti trice i „akužu“ kupa (as, dvica i trica kupa). Tom se igraču dodjeljuje 6 punata i ispisuje se na grafičkom sučelju koje on karte ima. Ovakav primjer implementacije „akužavanja“ onemogućava varanje i ne može se manipulirati s brojem karata u ruci.

„Akužavanje“ se vrši nakon što se izvrši dijeljenje karata unutar funkcije `DealCards`. Nakon što se sve karte podijele, napravi se provjera i pozove se funkcija `HasRequiedCards` i rezultat koji ona vrati (tipa `integer`) dodaje se timu kojem pripada igrač. Ako je korisnik ili „AIIgrač 2“ onda se bodovi dodjeljuju timu broj jedan a ako je „AIIgrač“ 1 i „AIIgrač 3“ onda se bodovi dodjeljuju timu broj 2, kako je prikazano na Ispisu 5.

```
int pointsToAdd = player.HasRequiredCards(message);
if (pointsToAdd > 0)
{
    if (player == players[0] || player==players[2])
        team1.AddPoints(pointsToAdd);
    if (player == players[1] || player==players[3])
    {
        team2.AddPoints(pointsToAdd);
    }
}
```

Ispis 5: Dodjela bodova akuže timovima.

Nakon što se podijele karte korisnik može početi igru. Korisnik počinje svaku rundu zbog toga što on pritišće dugme „DEAL CARDS“ i on se može smatrati djeliteljem . Korisnikovo igranje karte se odvija unutar funkcije `PlayYourCard` koja je tipa `IEnumerator`. U kontekstu Unityja, IEnumerator se često koristi za stvaranje korutina (coroutines), koje omogućavaju izvođenje funkcija u više koraka. Ovo je izuzetno korisno za implementaciju čekanja i vremenskih pauza unutar igre. Korutine u Unityju su posebne vrste funkcija koje omogućavaju programerima da zaustave izvršavanje funkcije na određeni vremenski period ili dok se ne ispuni određeni uvjet, a zatim nastave s izvršavanjem od točke gdje su zaustavljene. Ovo je vrlo korisno za animacije, čekanje određenog vremena, ili sekvencijalne događaje u igri. [12] Unutar funkcije `PlayYourCard` je ključno čekanje na korisnika da odigra kartu. Korisnik ima neograničeno vrijeme da odabere koju će kartu baciti i ima vremena razmisliti o svojoj strategiji i taktici. Tijekom tog vremena on može uputiti svoju poruku svom suigrači i sugerirati mu kartu koju će baciti, odnosno može koristiti dugmad za komunikaciju „Strišo“ i „Tučem“. Unutar ove funkcije se dodaju dugmad na karte tako da se, ako je došlo do greške (koja se može dogoditi zbog kompleksnosti razvojnog okruženja), svakoj karti doda njezina funkcionalnost i mogućnost da se ta karta odigra. Nakon postavljanja dugmadi na karte čeka se korisnik koji mora odigrati kartu uz pomoć funkcije koja je prikazana na Ispisu 6. Nakon korisnikove interakcije karte se prebacuju na stol i nakon toga se čekaju ostali igrači.

```
private IEnumerator playYourCard()
{

    foreach (GameObject card in players[0].hand)
    {
        AddButtonToCard(card);
    }

    yield return new WaitUntil(() => cardClickedBool && clickedCard != null);

    MoveCardToDesk(clickedCard, players[0]);

    cardClickedBool = false;
    clickedCard = null;




}
```

Ispis 6: Funkcija za odigravanje karte za korisnika

Igranje karata od strane računalnih igrača je osmišljeno unutar funkcije `PlayCardCoroutine` koja je isto tipa `IEnumerator`. Na početku ove funkcije poziva se pomoćna funkcija Unity okruženja `WaitForSeconds` koja je zadužena za vremensko čekanje ovisno o broju sekunda koje pošaljemo u funkciju. Nakon što prođe vrijeme koje je predviđeno za čekanje poziva se funkcija iz klase `PlayerPrefab` `ChooseCardToPlay` koja se nalazi na Ispisu 7. U tu se funkciju šalje je li početni potez kruga igre, ime igrača, karte koje se nalaze na stolu i stanje komunikacije, odnosno je li pritisnuto dugme strišo ili tučem. Ako je početni trenutak onda se šalje `true` ako nije onda se šalje `false`. Ovisno o tome, karte će biti bačene na stol.

```
private IEnumerator PlayCardCoroutine(PlayerPrefab player)
    {

        GameObject card;
        yield return new WaitForSeconds(1);
        if (player == players[starterID])
        {
            card = player.ChooseCardToPlay(deskGO.GetComponent<Desk>().cards, true,strisoBool,tucemBool);
        }
        else
        {

            card = player.ChooseCardToPlay(deskGO.GetComponent<Desk>().cards, false,strisoBool,tucemBool);

        }

        if (card != null)
        {
            MoveCardToDesk(card, player);
        }

    }
```

Ispis 7: Funkcija za odigravanje karte za računalnog igrača.

Nakon što igrači svi odigraju karte kreće se u odlučivanje tko je pobjednik kruga i kojem timu pripadaju bodovi. Funkcija za provjeru je prikazana u Ispisu 8. Ovaj dio kôda služi za određivanje pobjednika trenutnog kruga igre, dodjelu bodova pobjedničkom timu te ažuriranje stanja igre. Proces započinje inicijalizacijom varijabli `winnerID` i `highestCardValue`, koje se koriste za praćenje identifikacije pobjednika i najjače karte u trenutnom krugu. Zatim se određuje boja karata na stolu, što se uzima kao referenca za usporedbu snage karata.

Kroz petlju, koja prolazi kroz sve karte na stolu, uspoređuje se snaga svake karte s trenutno najjačom kartom koja odgovara traženoj boji. Ako je karta jača od trenutne najjače karte i pripada odgovarajućoj boji, ažuriraju se varijable `highestCardValue` i `winnerID` s novim vrijednostima.

Nakon što se utvrdi pobjednik kruga, provjerava se kojoj ekipi pobjednik pripada. Ako pobjednik pripada ekipi 1 (igrači s ID-jevima 0 ili 2), dodjeljuju se bodovi toj ekipi. Svaka karta na stolu doprinosi ukupnom broju bodova ekipe, a ako je trenutni krug posljednji (deseti krug), ekipa dobiva dodatan bod, odnosno 3 dodatne bele. Karte se tada premještaju u špil pobjedničke ekipe pomoću funkcije `MoveCardsToWinningTeamDeck`. Ta funkcija prebacuje karte na dvije hrpe na stolu na kojoj su skupljene sve karte koje je tim dobio .

Provjerava se je li ekipa 1 osvojila igru tako što je dosegla potrebni broj bodova (41). Ako jesu, bodovi obje ekipe se resetiraju i ispisuje se poruka da je ekipa 1 pobijedila. Ako pobjednik kruga nije iz ekipe 1, isti postupak se primjenjuje za ekipu 2. Bodovi se dodjeljuju ekipi 2, a ako je trenutni krug posljednji, ekipa također dobiva dodatan bod. Karte se premještaju u špil ekipe 2, te se provjerava jesu li osvojili igru.

Unutar ove aplikacije igra je implementirana tako da se pamti broj bela a ne bodova. Bodovi se prikazuju na radnoj površini ali se u pozadini spremaju samo bele. Broj bodova dobivamo jednostavnom računskom operacijom dijeljenja a bele računskom operacijom modula. Konačan rezultat se prikazuje na radnoj površini u okviru prikazanom na Slici 7. To je omogućene uz pomoć funkcije UpdatePointsGUI koja na radnoj površini unutar okvira prikazuje broj bodova i bela za svaki tim.

Slika 7: Traka za prikaz bodova.

Naposljetku, povijest odigranih karata se ažurira za svakog igrača. Svaka karta na stolu dodaje se u povijest karti svakog igrača, što omogućava igračima da prate koje su karte već odigrane i da planiraju svoje buduće poteze na temelju te informacije. Ovaj kôd omogućava glatku kontrolu toka igre i održava praćenje bodova i stanja svakog igrača te njihovih ekipa

```
int winnerID = 0;
        int highestCardValue = -1;

        Suit suit = desk.cards[0].GetComponent<Card>().getSuit();

        for (int i = 0; i < desk.cards.Count; i++)
        {
            GameObject cardGO = desk.cards[i];
            Card card = cardGO.GetComponent<Card>();
            if (card.strenght > highestCardValue && card.getSuit() == suit)
            {
                highestCardValue = card.strenght;
                winnerID = desk.CardPlayerTrownID[i];
            }
        }
        starterID = winnerID;

        if (winnerID == 0 || winnerID == 2)
        {
            Debug.Log("Team 1 wins");
            foreach (GameObject card in desk.cards)
            {
                team1.AddPoints(card.GetComponent<Card>().getPoints());
            }
            if (currentTurn == 9)
            {
                team1.AddPoints(3);
            }
            MoveCardsToWinningTeamDeck(desk.cards, team1, team1Deck);
            if (team1.CheckGameOver())
            {
                team1.points = 0;
                team2.points = 0;
                message.text = "Team 2 wins game with requied 41 points";

            }
        }
        else
        {
            foreach (GameObject card in desk.cards)
            {
                team2.AddPoints(card.GetComponent<Card>().getPoints());
            }
            if (currentTurn == 9)
            {
                team2.AddPoints(3);
            }
            MoveCardsToWinningTeamDeck(desk.cards, team2, team2Deck);
            if (team2.CheckGameOver())
            {
                team1.points = 0;
                team2.points = 0;
                message.text = "Team 2 wins game with requied 41 points";
            }
        }
        foreach(PlayerPrefab player in players)
        {
            for(int i = 0; i < 4; i++)
            {
                player.history.Add(desk.cards[i].GetComponent<Card>());
            }
        }



        desk.ClearDesk();
        if (team1.CheckGameOver()|| team2.CheckGameOver())
        {
            DealCards();
        }
        SimulateNextTurn();
        UpdatePointsGUI();

    }
```

Ispis 8: Funkcija za provjeru pobjednika

4.2. Računalno upravljani igrači

Ovo poglavlje objašnjava kako funkcionira rad računalno upravljanih igrača. Opisuje sve programske kodove koji su bitni za odabir karata i komunikaciju između igrača.

4.2.1. Struktura klase PlayerPrefab

Klasa PlayerPrefab centralni je dio implementacije igrača unutar igre Trešeta, koji uključuje upravljanje stanjem igrača, njegovom rukom karata, poviješću odigranih karata, te logikom za donošenje odluka o potezima. Ova klasa omogućuje implementaciju i ljudskih i računalnih igrača, pružajući osnovne funkcionalnosti potrebne za interakciju s igrom.

Klasa PlayerPrefab nasljeđuje MonoBehaviour, što omogućuje da se koristi unutar Unity okruženja kao komponenta pridružena određenom GameObject-u. Ova klasa definira nekoliko ključnih varijabli i metoda koje omogućuju upravljanje stanjem i ponašanjem igrača. Klasa je prikazana u Ispisu 9.

```
public class PlayerPrefab : MonoBehaviour
{
  public string playerName;
  public List<GameObject> hand;
  public GameObject LastPlayedCard;
  public List<Card> history;
  int count = 0;
}
```

Ispis 9: Struktura klase PlayerPrefab

Varijable klase igrač

1. **playerName** (tipa `string`): Ova varijabla čuva ime igrača. Ime igrača može biti korisnički definiran tekst za ljudskog igrača ili predefiniran tekst za računalnog igrača (AI).
2. **hand** (tipa `List<GameObject>`): Ova varijabla predstavlja ruku karata igrača. Svaka karta u ruci igrača je GameObject koji sadrži komponentu Card, koja čuva informacije o samoj karti, kao što su boja (suit) i vrijednost (value).

3. **LastPlayedCard** (tipa `GameObject`): Ova varijabla čuva referencu na posljednju kartu koju je igrač odigrao. Ovo je korisno za praćenje posljednjeg poteza igrača i može se koristiti za analizu strategije ili pravila igre.
4. **history** (tipa `List<Card>`) : Ova varijabla čuva povijest svih karata koje je igrač odigrao tijekom igre. Povijest omogućuje igračima i računalnim igračima da analiziraju prošle poteze i donose odluke na temelju prethodnih događaja u igri.
5. **count** (tipa `int`): Ova varijabla koristi se kao pomoćna varijabla za interno brojanje ili praćenje stanja unutar metoda klase. U primjeru kôda, koristi se za praćenje broja poteza unutar specifičnih logika odlučivanja.

U kontekstu igre Trešeta, igrač je entitet koji sudjeluje u igri s ciljem postizanja pobjede. Igrač može biti ljudski korisnik ili računalni (AI) igrač. Svaki igrač ima svoju ruku karata, povijest odigranih karata, i slijedi određena pravila igre za donošenje poteza.

Ljudski igrač je korisnik koji interaktivno sudjeluje u igri putem grafičkog sučelja. Ljudski igrač donosi odluke na temelju vlastite strategije, iskustva i razumijevanja igre. Implementacija ljudskog igrača u klasi `PlayerPrefab` omogućuje korisniku da vidi svoju ruku karata, bira karte za igru i prati povijest svojih poteza.

Računalni igrač, ili AI igrač, koristi algoritme i ad-hoc pravila za donošenje odluka tijekom igre. Računalni igrač analizira stanje igre, povijest poteza i trenutne karte na stolu kako bi donio optimalne odluke. Cilj je simulirati inteligentno ponašanje koje izaziva ljudske igrače i osigurava zanimljivu i izazovnu igru. Implementacija AI igrača koristi metode klase `PlayerPrefab` za analizu povijesti igre i donošenje strategijskih poteza.

4.2.2. Funkcionalnosti klase PlayerPrefab

Funkcija `ChooseCardToPlay` u klasi `PlayerPrefab` omogućuje računalnom igraču da odabere koju će kartu odigrati tijekom svog poteza. Ova funkcija uzima u obzir stanje igre, povijest poteza i trenutne karte na stolu kako bi donijela optimalnu odluku. Ova funkcija čini okosnicu umjetne inteligencije ove igre. Ona je glavni element kojim se osigurava taktičnost poteza računalnih igrača.

Prvi korak u funkciji je provjera ima li igrač karte u ruci. Ako nema karata, funkcija ispisuje poruku i vraća `null`, što signalizira da igrač ne može odigrati kartu. Ovo osigurava da igra ne pokuša odigrati potez bez dostupnih karata. Ovaj dio funkcije je osigurač da se funkcija ne pokreće ako je došlo do neke greške i prikazan je na Ispisu 10.

```
if (hand == null || hand.Count == 0)
        {
            Debug.Log($"{playerName} has no cards in hand.");
            return null;
        }
```

Ispis 10: Kontrolni dio kôda

Implementacija računalnog igrača u igri koristi ad-hoc pravila za analizu i donošenje odluka. Ove tehnike omogućuju računalnom igraču da se prilagodi dinamici igre i odgovori na poteze ljudskih igrača na inteligentan i strateški način. Time se postiže realistična simulacija igre koja pruža zadovoljavajuće iskustvo za sve sudionike. Računalni igrač postaje važan element igre, doprinosi dinamičnosti i interaktivnosti, te osigurava da svaki krug igre bude jedinstven i izazovan.

Sljedeći blok kôda se bavi specifičnim slučajem kada je ime igrača "AIPlayer2" i varijabla `striso` je postavljena na `true`. Ako postoje karte na stolu, funkcija identificira boju prve karte na stolu i traži sve karte iste boje u ruci igrača. Ako pronađe karte iste boje, odabire kartu s najmanjom snagom, postavlja je kao zadnju odigranu kartu (`LastPlayedCard`), uklanja je iz ruke i vraća je kao odabranu kartu. Ovaj dio funkcije omogućuje igraču da odgovori na prethodne poteze drugih igrača na inteligentan način i to je prikazano na Ispisu 11.

```
if (playerName == "AIPlayer2" && striso)
{
    if (cardsOnTable.Count > 0)
    {
        Suit startingSuit = cardsOnTable[0].GetComponent<Card>().getSuit();
        List<GameObject> sameSuitCards = hand.Where(cardGO => cardGO.GetComponent<Card>().getSuit() == startingSuit).ToList();

        if (sameSuitCards.Count > 0)
        {
            GameObject lowestStrengthSameSuitCard = sameSuitCards.OrderBy(cardGO => cardGO.GetComponent<Card>().getStrenght()).First();
            LastPlayedCard = lowestStrengthSameSuitCard;
            hand.Remove(lowestStrengthSameSuitCard);
            return LastPlayedCard;
        }
    }
}
```

Ispis 11: Odabir karte za igru kao je AIIgrač2 i varijabla „strišo“

Ako je `isStartingTurn` postavljeno na `true`, ime igrača je "AIPlayer2", i varijabla `tucem` je `true`, funkcija se ponaša drugačije. Provjerava je li ovo prvi potez (prvi potez u ovoj ruci) ,a ako jest, traži boju zadnje odigrane karte u povijesti i sve karte iste boje u ruci igrača. Ako pronađe karte iste boje, odabire kartu s najvećom snagom, postavlja je kao zadnju odigranu kartu, uklanja je iz ruke i vraća je kao odabranu kartu. Ovaj dio funkcije osigurava da igrač koristi strategiju zasnovanu na povijesti poteza.

Sljedeći blok kôda ponovno se bavi slučajem kada je ime igrača "AIPlayer2" i `tucem` je `true`. Ako postoje karte na stolu, funkcija identificira boju prve karte na stolu i traži sve karte iste boje u ruci igrača. Ako pronađe karte iste boje, traži kartu s najvećom snagom koja je veća od najveće snage karte iste boje na stolu. Ako pronađe takvu kartu, postavlja je kao zadnju odigranu kartu, uklanja je iz ruke i vraća je kao odabranu kartu. Ako ne pronađe kartu s većom snagom, odabire kartu s najmanjom snagom iste boje, postavlja je kao zadnju odigranu kartu, uklanja je iz ruke i vraća je kao odabranu kartu. Ako u ruci nema karte iste boje, odabire kartu s najmanjom snagom iz ruke, postavlja je kao zadnju odigranu kartu, uklanja je iz ruke i vraća je kao odabranu kartu. Ovaj dio funkcije omogućuje igraču da igra na temelju trenutnog stanja na stolu i prilagodi svoju strategiju.

```
if (isStartingTurn && playerName == "AIPlayer2" && tucem)
{
    if (count == 0)
    {
        count = 1;
        Suit lastSuitPlayed = history[history.Count - 3].cardSuit;
        List<GameObject> sameSuitCards = hand.Where(cardGO => cardGO.GetComponent<Card>().getSuit() == lastSuitPlayed).ToList();

        if (sameSuitCards.Count > 0)
        {
            GameObject strongestSameSuitCard = sameSuitCards.OrderByDescending(cardGO => cardGO.GetComponent<Card>().getStrenght()).First();
            LastPlayedCard = strongestSameSuitCard;
            hand.Remove(strongestSameSuitCard);
            return LastPlayedCard;
        }
    }
}
```

Ispis 12: Odabir karte za igru kao je AIIgrač2 i varijabla „tučem“

Ako je `isStartingTurn` postavljeno na `true`, funkcija koristi drugačiju logiku, odnosno ako nikakva komunikacija nije korištena ili ako potez počinje igrač iz protivničkog tima koristi se sljedeća logika. Kada igrač započinje potez, funkcija najprije analizira povijest odigranih karata. Kreira se rječnik koji prati koliko se puta svaka boja karata pojavila u povijesti poteza. Istodobno, kreira se i drugi rječnik koji prati boje karata u ruci igrača. Ova dva rječnika omogućuju funkciji da identificira boje koje su se pojavile ukupno deset puta, kombinirajući karte iz povijesti i karte u ruci. Ako takva boja postoji, funkcija pronalazi najsnažniju kartu te boje u ruci, postavlja je kao zadnju odigranu kartu, uklanja je iz ruke i vraća kao rezultat.

```
Dictionary<Suit, int> suitCountInHistory = new Dictionary<Suit, int>();
            foreach (Card card in history)
            {
                if (!suitCountInHistory.ContainsKey(card.cardSuit))
                    suitCountInHistory[card.cardSuit] = 0;
                suitCountInHistory[card.cardSuit]++;
            }

            Dictionary<Suit, int> suitCountInHand = new Dictionary<Suit, int>();
            foreach (GameObject cardGO in hand)
            {
                Card card = cardGO.GetComponent<Card>();
                if (!suitCountInHand.ContainsKey(card.cardSuit))
                    suitCountInHand[card.cardSuit] = 0;
                suitCountInHand[card.cardSuit]++;
            }

            foreach (Suit suit in System.Enum.GetValues(typeof(Suit)))
            {
                if (suitCountInHistory.ContainsKey(suit) && suitCountInHand.ContainsKey(suit) &&
                    suitCountInHistory[suit] + suitCountInHand[suit] == 10)
                {
                    List<GameObject> sameSuitCards = hand.Where(cardGO => cardGO.GetComponent<Card>().getSuit() == suit).ToList();
                    GameObject strongestSameSuitCard = sameSuitCards.OrderByDescending(cardGO => cardGO.GetComponent<Card>().getStrenght()).First();
                    LastPlayedCard = strongestSameSuitCard;
                    hand.Remove(strongestSameSuitCard);
                    return LastPlayedCard;
                }
            }
```

Ispis 13: Odabir najjače karte ako nema karata jedne boje

Ako nijedna boja ne zadovoljava uvjet od deset pojavljivanja, funkcija jednostavno bira najsnažniju kartu u ruci, postavlja je kao zadnju odigranu kartu, uklanja je iz ruke i vraća

kao rezultat. Ova strategija osigurava da igrač uvijek odigra najjaču moguću kartu kada nema drugih specifičnih smjernica.

Ako nije `isStartingTurn`, odnosno ako nije početni potez, funkcija koristi sličnu logiku kao i prethodno opisani slučaj kada je `tucem true`. Ako postoje karte na stolu, funkcija identificira boju prve karte na stolu i traži sve karte iste boje u ruci igrača. Ako pronađe karte iste boje, traži kartu s najvećom snagom koja je veća od najveće snage karte iste boje na stolu. Ako pronađe takvu kartu, postavlja je kao zadnju odigranu kartu, uklanja je iz ruke i vraća je kao odabranu kartu. Ako ne pronađe kartu s većom snagom, odabire kartu s najmanjom snagom iste boje, postavlja je kao zadnju odigranu kartu, uklanja je iz ruke i vraća je kao odabranu kartu. Ako u ruci nema karte iste boje, odabire kartu s najmanjom snagom iz ruke, postavlja je kao zadnju odigranu kartu, uklanja je iz ruke i vraća je kao odabranu kartu. Ovaj dio funkcije omogućuje igraču da igra na temelju trenutnog stanja na stolu i prilagodi svoju strategiju. Ovaj dio kôda je prikazan na Ispisu 14.

```
else
        {
            if (cardsOnTable.Count > 0)
            {
                Suit startingSuit = cardsOnTable[0].GetComponent<Card>().getSuit();
                List<GameObject> sameSuitCards = hand.Where(cardGO => cardGO.GetComponent<Card>().getSuit() ==
startingSuit).ToList();

                if (sameSuitCards.Count > 0)
                {
                    int highestSameSuitValue = cardsOnTable.Where(card => card.GetComponent<Card>().getSuit() ==
startingSuit).Max(card => card.GetComponent<Card>().getStrenght());
                    GameObject strongestSameSuitCard = sameSuitCards.Where(cardGO => cardGO.GetComponent<Card>
().getStrenght() > highestSameSuitValue).OrderByDescending(cardGO => cardGO.GetComponent<Card>
().getStrenght()).FirstOrDefault();

                    if (strongestSameSuitCard != null)
                    {
                        LastPlayedCard = strongestSameSuitCard;
                        hand.Remove(strongestSameSuitCard);
                        return LastPlayedCard;
                    }
                    else
                    {
                        GameObject lowestStrengthSameSuitCard = sameSuitCards.OrderBy(cardGO =>
cardGO.GetComponent<Card>().getStrenght()).First();
                        LastPlayedCard = lowestStrengthSameSuitCard;
                        hand.Remove(lowestStrengthSameSuitCard);
                        return LastPlayedCard;
                    }
                }
            }
            GameObject lowestStrengthCard = hand.OrderBy(cardGO => cardGO.GetComponent<Card>
().getStrenght()).First();
            LastPlayedCard = lowestStrengthCard;
            hand.Remove(lowestStrengthCard);
            return LastPlayedCard;
```

Ispis 14: Odabir najjače ili najslabije karte

Prilikom svakog kruga igre u listu se puni s odigranim kartama i time se prati stanje igre. Računalni igrači za razliku od korisnika imaju mogućnost brze i jednostavne analize liste koja im omogućuje siguran uspjeh prilikom igranja karte. Punjenje liste povijesti je prikazano u Ispisu 15.

```
for(int i = 0; i < 4; i++)
            {
                player.history.Add(desk.cards[i].GetComponent<Card>());
            }
```

Ispis 15: Punjenje liste povijesti

Računalni igrači koristeći brzu analizu karata koje su već bačene na stol i koje su već odigrane u prošlim potezima omogućuju računalu da ostvari uspjeh protiv iskusnih igrača. Primjenjuju se ad-hoc pravila koja daju dojam da računala koriste umjetnu inteligenciju i mehaničko učenje ali se dokazano koriste samo jednostavni algoritmi i iteracije za provjeru prethodnog stanja. Svaki se potez sprema u memoriji i time se osigurava brza primjena ad-hoc pravila.

4.2.3. Implementacija Tučem i Strišo

Tučem je grafički implementiran uz pomoć dugmeta „Tučem“ koje se nalazi na grafičkom sučelju. Na tome dugmetu je postavljena mogućnost „slušanja“(*Lisener*) odnosno kada se pritisne ovo dugme onda se dogodi reakcija u pozadini. Ta reakcija postavlja `bool` varijablu `tucemBool` na `true` (istina). Ta se varijabla koristi unutar aplikacije i ako se uspješno izvrši odnosno ako se svi uvjeti poklope onda se ta varijabla 2 kruga nakon postavlja na početno stanje odnosno `false`. Stanje koje se mora poklopiti je to da korisnik ima odgovarajuće karte i da mu njegov suigrač može odgovoriti s kvalitetnim kartama koje garantiraju pobjedu kruga. Nakon što se krug izvrši uspješno sljedeći potez započinje suigrač odnosno AIIgrač 2 i on baca kartu iste boje kojeg mu je prethodno sugerirao korisnik. Karta koje će biti bačena na početku sljedećeg kruga biti će boje koja je u prethodnom krugu bila glavna. To će program saznati korištenjem liste `history` u kojoj se nalazi povijest svih karata koje su odigrane. Igrač će baciti boju karte koja je za četiri mjesta udaljena od kraja liste. Implementacije je prikazana na Ispisu 16.

```
if (playerName == "AIPlayer2" && tucem)
        {
            if (cardsOnTable.Count > 0)
            {
                Suit startingSuit = cardsOnTable[0].GetComponent<Card>().getSuit();
                List<GameObject> sameSuitCards = hand.Where(cardGO => cardGO.GetComponent<Card>().getSuit() == startingSuit).ToList();

                if (sameSuitCards.Count > 0)
                {
                    int highestSameSuitValue = cardsOnTable.Where(card => card.GetComponent<Card>().getSuit() == startingSuit).Max(card => card.GetComponent<Card>
().getStrenght());
                    GameObject strongestSameSuitCard = sameSuitCards.Where(cardGO => cardGO.GetComponent<Card>().getStrenght() >
highestSameSuitValue).OrderByDescending(cardGO => cardGO.GetComponent<Card>().getStrenght()).FirstOrDefault();

                    if (strongestSameSuitCard != null)
                    {
                        LastPlayedCard = strongestSameSuitCard;
                        hand.Remove(strongestSameSuitCard);
                        return LastPlayedCard;
                    }
                    else
                    {
                        GameObject lowestStrengthSameSuitCard = sameSuitCards.OrderBy(cardGO => cardGO.GetComponent<Card>().getStrenght()).First();
                        LastPlayedCard = lowestStrengthSameSuitCard;
                        hand.Remove(lowestStrengthSameSuitCard);
                        return LastPlayedCard;
                    }
                }
            }
            GameObject lowestStrengthCard = hand.OrderBy(cardGO => cardGO.GetComponent<Card>().getStrenght()).First();
            LastPlayedCard = lowestStrengthCard;
            hand.Remove(lowestStrengthCard);
            return LastPlayedCard;
        }
```

Ispis 16: Odabir karte za tučem.

Strišo je implementiran slično kao i tučem. Na radnoj površini postoji dugme koje na pritisak pokreće radnju u pozadini da suigrač može prepoznati njegov potez. Nakon što korisnik pritisne to dugme i odigra kartu suigrač neće po standardnoj proceduri za igru baciti najjaču kartu te boje, odnosno neće „ubiti“ kartu veće će baciti slabiju kartu. Pritiskom na dugme „Tučem“ `bool` varijabla `tucemBool` koja se unutar funkcije `hasRequiedCards` koristi za odabir karte. Ova se varijabla, za razliku od slučaja s tučem, odmah nakon što se odigra karta postavlja na `False` . Strišo je implementiran uz pomoć kôda koje se nalazi na ispisu 17.

```
if (playerName == "AIPlayer2" && striso)
        {
            if (cardsOnTable.Count > 0)
            {
                Suit startingSuit = cardsOnTable[0].GetComponent<Card>().getSuit();
                List<GameObject> sameSuitCards = hand.Where(cardGO => cardGO.GetComponent<Card>().getSuit() == startingSuit).ToList();

                if (sameSuitCards.Count > 0)
                {
                    GameObject lowestStrengthSameSuitCard = sameSuitCards.OrderBy(cardGO => cardGO.GetComponent<Card>().getStrenght()).First();
                    LastPlayedCard = lowestStrengthSameSuitCard;
                    hand.Remove(lowestStrengthSameSuitCard);
                    return LastPlayedCard;
                }
            }
        }
```

Ispis 17: Odabir karte za strišo

5. Zaključak

Izrada igre Trešeta u Unity okruženju predstavljala je izazovan i složen projekt koji je uspješno realiziran kroz niz faza uključujući dizajn, implementaciju, testiranje i optimizaciju. Kroz ovaj rad prikazana je cjelovita implementacija igre koja obuhvaća intuitivno i pregledno 2D grafičko sučelje, sofisticirane i napredne algoritme za računalne igrače, detaljno kodirana pravila igre, sustav igre u parovima te mehanizam za praćenje bodova kroz više partija.

Glavni doprinos ovog rada leži u stvaranju autentičnog i izazovnog iskustva kartaške igre Trešeta, koje vjerno odražava tradicionalnu verziju igre, dok istovremeno koristi napredne tehnike primjene ad-hoc pravila kako bi osigurala realističnu i dinamičnu igru. Kroz pamćenje povijesti poteza i donošenje odluka temeljenih na analizi prethodnih akcija, računalni igrači pružaju korisniku izazovnog protivnika koji simulira ljudske strategije.

Sustav igre u parovima omogućava suradnju i koordinaciju strategija među igračima, dodatno obogaćujući iskustvo igre. Pamćenje bodova kroz više partija osigurava dugotrajnu kompetitivnost i praćenje napretka, što motivira igrače da pažljivo biraju poteze jer se svaki prati, a računalni igrač može iskoristiti njihove greške i uzrokovati gubitak bodova.

Tijekom razvoja igre posebna pažnja posvećena implementaciji pravila kako bi se osigurale jednake šanse za sve igrače, što je ključno za održavanje interesantnosti i pravednosti igre. Dizajn korisničkog sučelja, koji je intuitivan i pregledan, omogućava igračima lako razumijevanje stanja igre i donošenje informiranih odluka.

Primjeri kodova i algoritama rješenja detaljno su opisani i mogu poslužiti kao smjernice za razvojne programere koji žele kreirati slične projekte.

Ovaj rad doprinosi polju razvoja računalnih igara kroz detaljan prikaz procesa stvaranja složene igre u Unity okruženju. Također, pruža korisne smjernice za buduće projekte, nudeći čvrstu osnovu za daljnji razvoj i unapređenje digitalnih kartaških igara temeljenih na tradicionalnim pravilima.

6. Literatura

[1]Wikipedia – Trešeta - https://hr.wikipedia.org/wiki/Tre%C5%A1eta - (Posjećeno 24.7.2024.)

[2] Unity shooter game tutorial - http://xrayisgray.de/sites/tutorial_3.html (Posjećeno 29.07.2024.)

[3] Wikipedia – „Unity (game engine)“ - https://en.wikipedia.org/wiki/Unity_(game_engine) - (Posjećeno 11.07.2024.)

[4] Unity Asset Store – FAQ -https://support.unity.com/hc/en-us/categories/201253946-Asset-Store - (Posjećeno 13.07.2024)

[5]Wikipedia – Briškula - https://hr.wikipedia.org/wiki/Bri%C5%A1kula - (Posjećeno 13.7.2024.)

[6] „Unity Documentation“ – „Buttons“ - https://docs.unity3d.com/2018.2/Documentation/ScriptReference/UI.Button.htm - (posjećeno 12.07.2024)

[7] „Unity“ – „Canvas“ https://docs.unity3d.com/Packages/com.unity.ugui@1.0/manual/UICanvas.html - (Posjećeno 12.07.2024)

[8] Unity Asset Store Manual - https://docs.unity3d.com/es/2019.3/Manual/AssetStore.html (Posjećeno 29.07.2024.)

[9] Unity Documentation – Unity Assets Workflow https://docs.unity3d.com/560/Documentation/Manual/AssetWorkflow.html - (Posjećeno 17.07.2024)

[10] Unity Manual – Unity inspector . https://docs.unity3d.com/510/Documentation/Manual/Inspector.html (Posjećeno 19.08.2024.)

[11] Unity Documentation – Scripting Images – https://docs.unity3d.com/2018.2/Documentation/ScriptReference/UI.Image.html - (Posjećeno 18.07.2024)

[12] Unity Manual – Coroutines - https://docs.unity3d.com/Manual/Coroutines.html (Posijećeno 29.7.2024.)