SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 1207

3D računalna igra inspirirana društvenim igrama

Ela Marušić

Zagreb, lipanj 2016.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA
ODBOR ZA DIPLOMSKI RAD PROFILA

Zagreb, 8. ožujka 2016.

DIPLOMSKI ZADATAK br. 1207

Pristupnik: **Ela Marušić (0036469019)**
Studij: Računarstvo
Profil: Računarska znanost

Zadatak: **3D računalna igra inspirirana društvenim igrama**

Opis zadatka:

Cilj diplomskog rada je osmisliti i u potpunosti implementirati stratešku računalnu igru inspiriranu modernim društvenim igrama. Za ostvarenje zadatka potrebno je osmisliti i implementirati mehaniku i pravila igre.
Kako bi igra bila zanimljivija, predviđeno je da se igraća ploča automatski generira te je podijeljena na mrežu polja po kojima se igrači mogu kretati. Generiranje ploče treba ostvariti na način da su svi igrači na početku u ravnopravnoj poziciji. Predviđena je igra za dva ili više igrača u umreženom načinu rada.
Vaša je zadaća proučiti dostupne sustave za razvoj igara, osmisliti igru i u potpunosti je implementirati.
Svu potrebnu literaturu i uvjete za rad osigurat će Vam Zavod za telekomunikacije.

Zadatak uručen pristupniku: 18. ožujka 2016.
Rok za predaju rada: 1. srpnja 2016.

Mentor:

Prof. dr. sc. Igor Sunday Pandžić

Djelovođa:

Doc. dr. sc. Tomislav Hrkać

Predsjednik odbora za diplomski rad profila:

Prof. dr. sc. Siniša Srbljić

Sadržaj

Uvod

Pojam „društvena igra“ najčešće se odnosi na igre za više igrača koje se igraju prema određenim pravilima pomoću ploče, figura, kocaka ili karata. Postoji mnogo različitih žanrova ovakvih igara te se ishod igre kod nekih više temelji na sreći (npr. Čovječe, ne ljuti se), a kod nekih na strategiji (npr. šah). Uz poznate tradicionalne igre kao što su šah i Monopoly posljednjih godina izdalo se mnoštvo novih, a popularnije igre ostvarile su i svoje računalne kopije (npr. brojne inačice šaha i raznih kartaških igara, Monopoly, Settlers of Catan). Tipične mehanike i pravila ovakvih igara inspirirali su i mnoge originalne video igre koje nisu direktna preinaka. Poznati i uspješni primjeri su strateška igra Armello te kartaška igra Hearthstone. Glavna prednost računalnih igara nad fizičkim je što se igrači ne moraju nalaziti u istoj prostoriji, već se mogu igrati preko interneta te ljudske igrače može zamijeniti računalo.

Tema ovog rada izrada je računalne igre inspirirane mehanikama igara na ploči, s fokusom na strateško igranje. U prvog poglavlju rada navedeni su zahtjevi zadatka. Drugo poglavlje opisuje prijedlog rješenja, podijeljeno na prijedlog dizajna igre i prijedlog tehnologija koje će se koristiti. Treće poglavlje fokusira se na implementaciju igre, s poglavljima koja objašnjavaju izvedbu igraćih karata, generiranja ploče i umrežavanja pomoću pokretačkog sustava Unity. U privitku se nalaze upute za pokretanje igre.

1. Specifikacija zahtjeva

Zadatak je osmisliti pravila za društvenu igru na ploči te ostvariti njenu digitalnu implementaciju. Igra je namijenjena za dva ili više igrača koji se figurama kreću po ploči. Ploča igre treba biti podijeljena na mrežu polja koja imaju različite učinke na igrača koji ih zauzme. Raspored takvih polja generirat će se računalom tako da je ploča drukčija pri svakom igranju, no niti jedan igrač na početku igre nije na takvoj poziciji da ima značajnu prednost nad ostalima. Svakom je igraču samo tijekom njegovog poteza omogućeno izvršavanje važećih akcija te igra završava kad jedan od igrača dostigne definirani uvjet za pobjedu. Igračevo donošenje odluka o akcijama koje će izvršiti tijekom svog poteza igri treba dati strateški element – odnosno, mehanike igre ne smiju biti takve da ishod igre ovisi isključivo o sreći, nego takve da igračeva vještina u igranju ima utjecaj na ishod.

Implementirana igra mora imati mogućnost umreženog načina rada. Igrači se moraju moći spojiti preko interneta te tako zajedno igrati, svatko na svom računalu.

2. Prijedlog rješenja

2.1. Prijedlog pravila i mehanika

Predlaže se igra za dva do četiri igrača. Igra se u potezima i igrači počinju na predefiniranim poljima na ploči, ovisno o broju igrača. Igračima su omogućena 3 koraka po potezu i mogu se pomicati samo u susjedna polja. Većina polja nema efekt, a najvažnija polja su naselja. Kad igrač stane na polje koje je naselje, on ga osvoji. Osvajanje naselja donosi dva zlatnika i vlasništvo nad tim naseljem. Nakon svakog kruga, svaki igrač dobije po jedan zlatnik za svako naselje koje je u njegovom vlasništvu. Naselja koja pripadaju drugim igračima mogu se osvojiti.

Kao dodatna mehanika koja će dodati više strategije u igru predlažu se igraće karte. Igrači će započeti igru s tri karte u ruci te će na početku svakog svog poteza povući još jednu. Karte se mogu odigrati samo tijekom vlastitog poteza. Karte mogu utjecati ili na polja na ploči ili na igrače te mogu imati pozitivan ili negativan efekt. Nekoliko primjera:

- igrač koji odigra kartu dobije dva dodatna koraka za kretanje
- igrač koji odigra kartu bira naselje koje je od sad u njegovom vlasništvu
- igrač koji odigra kartu bira igrača (može i sebe) koji će se vratiti na poziciju u kojoj je započeo igru

Igra završava kad jedan od igrača skupi 25 (ili više) zlatnika te se taj igrač proglašava pobjednikom.

2.2. Prijedlog tehničke izvedbe

Za izradu video igre s opisanim mehanikama koristit će se pokretački sustav Unity. Unity je besplatni pokretački sustav od tvrtke Unity Technologies koji omogućava razvoj 3D i 2D aplikacija za razne platforme (uz Windows, Linux i Mac podržava i razne mobilne platforme i konzole) u programskim jezicima C#, UnityScript i Boo. Za razvoj ove igre koristit će se jezik C# i platforma Windows.

Unityjev UNET sustav (skraćenica za Unity Networking) koristit će se za ostvarenje umreženog igranja. UNET nudi aplikacijsko programsko sučelje (engl. *application programming interface*, API) visoke i niske razine, a za potrebe ovog rada dostatno je sučelje visoke razine koje implementira jednostavan klijent-poslužitelj (engl. *client-server*) mrežni model.

Naposljetku, igra će sadržavati 3D modele koji predstavljaju polja na ploči te figure igrača. Za izradu tih modela koristit će se program za 3D modeliranje Blender.

3. Programska izvedba

3.1. Igraće karte

Slike karata koje su igraču dostupne iscrtane su na ekranu. Korisnik ih može mišem povlačiti po ekranu i pokušati odigrati kartu otpuštanjem tipke. Karta će tad biti uspješno odigrana ovisno o karti i poziciji miša. Karta koja ima efekt na igrače mora biti odigrana iznad polja na kojem stoji neki igrač, a karta koja ima efekt na polje mora biti odigrana iznad odgovarajućeg tipa polja. Ako je karta uspješno odigrana, njen se efekt izvrši na svim klijentima.

Karte su implementirane pomoću nasljeđivanja. Osnovna klasa `CardInfo` sadrži:

- varijable `id`, `cardName`, `description` i `copiesInDeck` koje sadrže informacije o imenu i opisu karte te broj kopija te karte u špilu.
- virtualne funkcije `CardPlayedAt (Vector3 pos)` i `PlayCardEffect (int targetId, Player player)`. Prva funkcija poziva se na klijentu kad korisnik pokuša odigrati kartu te vraća `true` ako se karta može odigrati na toj poziciji (`false` inače). Druga se funkcija poziva na svim klijentima ako je jedan od igrača uspješno odigrao tu kartu. Argumenti su ID polja nad kojim je odigrana karta i igrač koji ju je odigrao.

Primjer ovakve implementacija je klasa `Card_OvertakeVillage` koja igraču dodijeli vlasništvo nad odabranim naseljem. Funkcija `CardPlayedAt` pomoću Unityjeve klase `Raycast` sazna koji se objekt nalazi ispod pokazivača miša. Zatim sazna je li objekt naselje tako da provjeri ima li objekt komponentu `VillageTile`. Funkcija `PlayCardEffect` kao argumente dobije ID naselja i referencu na novog vlasnika pa jednostavno izvrši svoj efekt:

```
VillageTile target =
(VillageTile)board.GetTileWithId(targetId);
target.ChangeOwner(player);
```

Špil karata implementiran je klasom `Deck` koja sadrži listu `CardInfo` referenci i funkcije `Shuffle()` i `DrawCard()`. Lista se na početku igre napuni tako da se svaki tip

karte ubaci onoliko puta koliko je definirano varijablom `copiesInDeck`, zatim se lista promiješa funkcijom `Shuffle()`. `DrawCard()` vraća kartu koja se nalazi na vrhu liste i izbriše je.

`Shuffle()` implementira Fisher-Yates algoritam za generiranje slučajne permutacije konačnog seta. Ideja je da se iz niza uzima slučajan element i prebacuje u novi niz dok ne prebacimo sve elemente i tako dobijemo novi, permutirani niz. Da bi algoritam imao složenost O(n) potrebna je mala preinaka: da se izbjegne operacija brisanja elemenata, odabrani element će se zamijeniti sa zadnjim elementom originalnog niza. Tako se permutirani niz gradi od kraja, a elementi biraju od početka. [4] Kod je sljedeći:

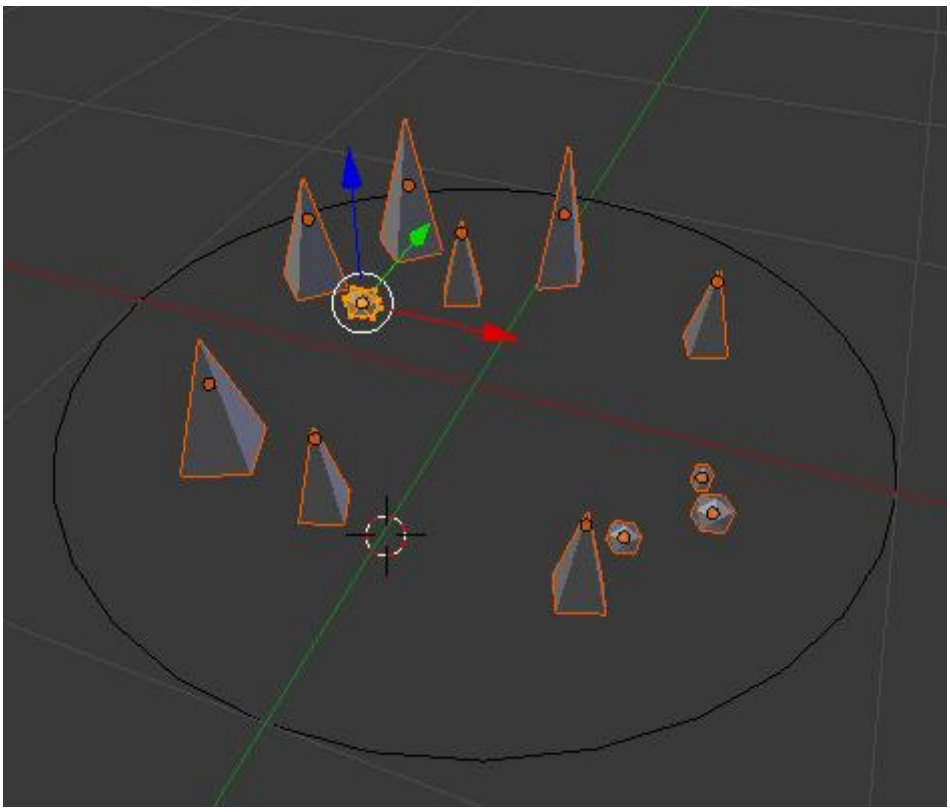
```
public void Shuffle<T>(List<T> array)
{
	int count = array.Count;
	for (int i = count - 1; i > 0; --i) {
		int randIndex = Random.Range(0, i);

		T temp = array[i];
		array[i] = array[randIndex];
		array[randIndex] = temp;
	}
}
```

3.2. Generiranje ploče

Ploča je implementirana pomoću nekoliko klasa. Polja ploče imaju sličnu izvedbu kao i karte. Osnovna klasa `Tile` sadrži varijable i funkcije koje su zajedničke svim poljima, poput funkcija za promjenu boje okvira čime se korisniku naznačuje da na to polje može pomaknuti svoju figuru i `PlayEffect(Player player)` funkcije koja se pozove kad igrač zauzme polje. Posebne vrste polja nasljeđuju Tile i dodaju svoju specifičnu funkcionalnost. Npr. klasa `VillageTile` implementira naselje koje daje zlatnike. Ona ima svoju verziju `PlayEffect` funkcije koja promijeni vlasnika tog polja i dodijeli mu zlatnike. Uz to ima i `AwardGold()` funkciju koja se poziva na kraju kruga te dodijeli vlasniku dodatni zlatnik.

`BoardManager` klasa sadrži liste referenci na razne vrste polja i funkcije vezane za funkcionalnost ploče tijekom igre poput dohvaćanja polja koje ima određeni ID, dohvaćanja susjednih polja, i slično.

Klasa `BoardGenerator` sadrži funkcionalnost vezanu uz generiranje ploče i varijable s informacijama o ploči koje su potrebne pri generiranju poput dimenzija ploče i početnih pozicija igrača. Glavna funkcija je funkcija `Generate()` čijim se pozivom započne generiranje. Prvo se instanciraju posebna početna polja igrača čije su lokacije predodređene. Vrstu ostalih polja određuje se pomoću generatora slučajnih brojeva. Nakon instanciranja modela za polja, na njih se primjeni slučajna rotacija kako bi ploča izgledala prirodnije. Modeli su pažljivo modelirani unutar kružnice upisane u okvir polja (heksagon) kako bi se mogli rotirati bez da izađu iz okvira (Slika 3.1).



Slika 3.1 Prikaz modela polja unutar Blendera

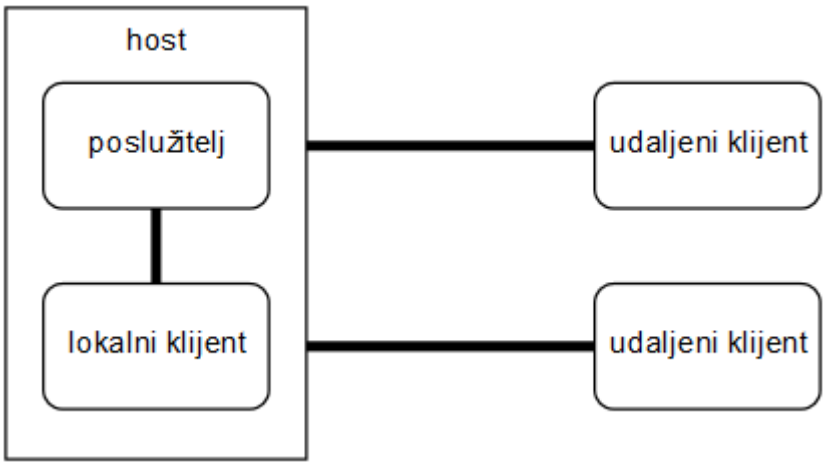
Da igra ne bi bila nepošteno balansirana u korist nekog od igrača, određena polja koja su najbliža početnim poljima igrača čine zonu igrača te se pri generiranju pazi da imaju sličan broj važnih polja poput naselja. Ostala polja spadaju u neutralnu zonu i mogu imati bilo kakav raspored tipova polja.

3.3. Umreženi način rada

3.3.1. UNET (HLAPI)

Unityjev sustav za umrežavanje sadrži API niske razine (engl. *low level API,* LLAPI) i API visoke razine (engl. *high level API,* HLAPI). LLAPI je namijenjen izradi igara sa specifičnim potrebama koje ne zadovoljava Unityjeva implementacija i korisnicima koji žele izraditi vlastitu mrežnu infrastrukturu. Kako za potrebe ovog rada korištenje LLAPI nije bilo potrebno, daljnji tekst fokusirat će se na HLAPI rješenje.

Način korištenja i mogućnosti HLAPI-ja opisani su u [1] i [2], a izvorni kod dostupan je na [3]. HLAPI implementira klijent-poslužitelj model prikazan na slici (Slika 3.2). Sva se komunikacija između umreženih računala odvija preko poslužitelja. Poslužitelj može biti i domaćin (engl. *host*) - kombinacija klijenta i poslužitelja. Takav se klijent vodi kao lokalni klijent te dijeli proces s poslužiteljem. Zbog toga se poruke između poslužitelja i lokalnog klijenta ne razmjenjuju preko interneta već pomoću redova poruka (engl. *message queue*). Treba napomenuti da je ovo interna razlika pa se HLAPI kod piše neovisno o vrsti klijenta. Ovo omogućuje jednostavniji razvoj igara koje nude opcije samostalnog i mrežnog igranja jer ne treba pisati odvojene skripte za svaki način rada.



Slika 3.2 Mrežni model s klijentima i *hostom*

Klasa `NetworkManager` sadrži funkcionalnost vezanu za stanje mreže. Značajnije varijable i funkcije koje sadrži su:

- funkcije `StartHost()`, `StartClient()` i `StartServer()` koje se pozivaju ovisno o tome želi li korisnik započeti igru kao *host*, klijent ili običan poslužitelj.

- varijable `networkAddress` i `networkPort`, tj. IP adresa i port koji će se koristiti za uspostavu veze.
- varijable `offlineScene` i `onlineScene` koje sadrže ime scene u kojoj se korisnik nalazi prije uspostave veze i scene u koju se prebacuje nakon poziva jedne od gornje navedenih funkcija. Prva scena obično sadrži izbornik, a druga je obično scena u kojoj započinje igra. Ako dođe do raskida veze, igra se vraća u prvu scenu.
- varijabla `playerPrefab` (prefab u Unityju je tip objekta s predefiniranim skriptama i ostalim komponentama) koja sadrži referencu na objekt koji u igri predstavlja igrača. Taj će se objekt instancirati za svakog klijenta koji se spoji.

Opisano ponašanje dijelom je nepoželjno za vrstu igre koja se izrađuje u ovom radu. Igrači nakon uspostave veze ne smiju odmah biti ubačeni u igru, već bi igra trebala početi nakon što svi spojeni klijenti naznače da su spremni za početak. Do tad bi se igrači trebali nalaziti u sobi (engl. *lobby*) u kojoj mogu namještati postavke (npr. promijeniti svoje ime i boju figure u igri) i pričekati ostale sudionike. *Lobby* u igru možemo dodati korištenjem klase `NetworkLobbyManager` koja nasljeđuje `NetworkManager` te dodaje sljedeće:

- varijabla `LobbyPlayerPrefab` koja ima slično ponašanje kao `playerPrefab`. Instancira se kad se klijent spoji te sadrži informacije o tom igraču poput imena i boje te je li igrač spreman za početak igre. `PlayerPrefab` se sad instancira za svakog igrača tek kad igra započne.
- varijabla `minimumPlayers` koja određuje koji je najmanji broj spremnih igrača dovoljan za početak igre. Igra u svakom slučaju neće početi dok svi spojeni klijenti nisu spremni, no neće početi ni ako su svi klijenti spremni, a manje ih je nego vrijednost spremljena `u minimumPlayers`.
- `maxPlayers` koja određuje najveći broj klijenata koji se mogu spojiti u *lobby*.
- `lobbyScene` i `playScene` koje su zamijenile `offlineScene` i `gameScene`. Razlika je što se u `playScene` (scenu igre) ulazi tek kad su zadovoljeni uvjeti za početak igre.
- funkcija `OnLobbyServerSceneLoadedForPlayer (GameObject lobbyPlayer, GameObject gamePlayer)` poziva se na poslužitelju (ili *hostu*) kad na nekom klijentu završi učitavanje scene igre. Argumenti su objekti koji predstavljaju tog igrača u *lobbyju* i u igri te bi se ovdje trebalo obaviti

primjenjivanje postavki koje je igrač postavio u *lobbyju* na objekt koji ga predstavlja u igri.

`NetworkLobbyManager` klasom ostvarena je većina funkcionalnosti umreženog igranja, no još treba omogućiti razmjenu informacija tijekom same igre. HLAPI nudi više načina za slanje poruka preko mreže:

- Sinkronizacija stanja pomoću `[SyncVar]` atributa. Vrijednost varijable uz koju stoji taj atribut automatski će se poslati svim klijentima ako joj se vrijednost promijeni na poslužitelju. Treba imati na umu da se sinkronizacija ne vrši u obrnutom smjeru (od klijenta prema serveru) i da `[SyncVar]` može stajati samo uz osnovne tipove varijabli (npr. string, integer, float) i strukture.
- Slično sinkronizaciji varijabli, moguća je i sinkronizacija listi pomoću posebnih tipova podataka `SyncListInt`, `SyncListFloat`, `SyncListString`, `SyncListUInt`, `SyncListBool` i `SyncListStruct`.
- Udaljeni pozivi funkcija koji su mogući u oba smjera. Funkcije koje se pozivaju na poslužitelju, a izvršavaju na klijentima, označavaju se atributom `[ClientRpc]` te imaju ime s prefiksom „`Rpc`“. Slično tome, funkcije koje se pozivaju na klijentu, a izvršavaju na poslužitelju, imaju atribut `[Command]` i ime s prefiksom „`Cmd`“. Pozivanje `Command` funkcije dozvoljeno je samo na objektu igrača koji pripada tom klijentu (takozvani lokalni igrač, ostali objekti igrača pripadaju udaljenim klijentima). Argumenti funkcija se također šalju preko mreže ako se mogu serijalizirati. Ovaj se način razmjene poruka najviše koristio u razvoju igre.

Način razmjene poruka koji se najviše koristio u razvoju igre su udaljeni pozivi funkcija. Sva mrežna logika koja je specifična za igru nalazi se u dvije klase: `Player` i `GameLogic`.

3.3.2. Skripta Player

Skripta `Player` prikvačena je za objekt igrača koji se instancira nakon što se učita scena igre. Nasljeđuje klasu `NetworkBehaviour` da bi se u kodu mogle koristiti `Command` i `ClientRpc` funkcije te da bi imala pristup varijablama koje sadrže informacije vezane uz

mrežu, poput varijable `isLocalPlayer` prema kojoj se zna je li objekt na kojoj se skripta nalazi lokalni igrač.

Klasa `Player` sadrži nekoliko `SyncVar` varijabli: boju, ime, ID i sjeme za generator slučajnih brojeva (engl. *random number generator seed*, RNG *seed*). Njihove vrijednosti postavlja `NetworkLobbyManager` prema vrijednostima odgovarajućeg *lobby* igrača. Svrha sinkronizacije sjemena je da klijenti generiraju identične slučajne brojeve. Tako pri generaciji ploče poslužitelj ne mora klijentima slati raspored i vrste polja jer će klijenti jednostavno generirati identične ploče.

Ostale varijable su većinom reference na razne objekte u sceni te informacije o igraču poput broja karata u ruci, preostalih koraka za kretanje, trenutnog polja, etc. Njihove se vrijednosti ne sinkroniziraju preko mreže automatski, već se oslanja na pozive udaljenih funkcija. Ideja je da klijent `Command` funkcijom *hostu* pošalje informaciju o akciji koju želi napraviti te *host* `ClientRpc` funkcijom svim klijentima javi da obave određenu akciju za tog igrača. Posljedica toga je da će na svim klijentima vrijednosti takvih varijabli biti ažurirane. Primjer toga je funkcija `RpcAddCard`:

```
[ClientRpc]
public void RpcAddCard(int id)
{
	handSize++;
	if (!isLocalPlayer)
		return;
	UI.AddCard(id);
}
```

Kad neki igrač vuče kartu, *host* poziva navedenu funkcija te se ona izvršava na svim klijentima. Svi klijenti vide da se tom igraču povećao broj karata u ruci, a samo se klijentu koji upravlja tim igračem prikaže nova karta na ekranu.

Klasa `Player` surađuje s `GameLogic` da bi se igra uspješno inicijalizirala nakon što se učita scena igre. Na ekranu je za to vrijeme prikazana slika za učitavanje (engl. *loading screen*). Slijedi opis značajnijih funkcija:

- Funkciju `Initialize()` poziva `GameLogic` kad se u sceni klijenta instanciraju svi igrači. U njoj se poziva generacija ploče i inicijalizacija klase za korisničko sučelje (engl. *user interface*, UI) te se onda poziva *Command* `CmdClientReady()`. Ta se funkcija ne izvršava na tom klijentu, nego na *hostu*.

Tako klijent *hostu* daje do znanja da je njegova scena inicijalizirana i da je spreman za početak igre.

- `RpcStartTurn()` funkcija pozove se kad započne potez tog igrača. Treba napomenuti da se funkcija izvršava na svim klijentima, što je poželjno jer na svim računalima treba prikazati prozor s informacijom o promjeni trenutnog igrača i obavijestiti kameru da prati njegovu poziciju. Ako je vrijednost varijable `isLocalPlayer` istinita (dakle, ako se kod vrti na klijentu koji upravlja tim igračem), omogućit će se slušanje ulaza (engl. *input*). Kad korisnik klikne na gumb za završetak poteza, poziva se `CmdEndTurn()` te se time javlja *hostu* da je klijent gotov s potezom.
- Ako korisnik odigra kartu, redom se pozivaju funkcije `PlayCard`, `CmdPlayCard`, `RpcPlayCard`. Ovako klijent na kojem je odigrana karta javlja *hostu* koja je karta odigrana i na što (ili koga), *host* to proslijedi svim klijentima te se efekt karte izvrši na svim klijentima. Ideja je ista i za pomicanje igrača po ploči.

```
public void PlayCard(CardInfo card, int targetId)
{
      CmdPlayCard(card.id, targetId);
}

[Command]
public void CmdPlayCard(int cardId, int targetId)
{
      RpcPlayCard(cardId, targetId);
}

[ClientRpc]
public void RpcPlayCard(int cardId, int targetId)
{
      handSize--;
      CardInfo card = deck.GetCardWithId(cardId);
      card.PlayCardEffect(targetId, this);
}
```

3.3.3. Skripta GameLogic

`GameLogic` implementira logiku igre i potrebnu mrežnu funkcionalnost (zbog čega, kao i `Player`, nasljeđuje `NetworkBehaviour`). Sadrži listu igrača (`Player` instanci) i varijable vezane za pravila igre poput maksimalnog broja karata u ruci, broja karata koje igrači dobiju na početku igre, broja zlatnika koji je potreban za pobjedu, etc.

Za elegantniju implementaciju toka igre, korištene su korutine (engl. *coroutines*). Obične funkcije u potpunosti se izvrše prije nego se iscrta slika na ekranu, a korutine su funkcije čije se izvršavanje može zaustaviti i zatim nastaviti nakon što se ispuni neki uvjet. Korutina se poziva funkcijom `StartCoroutine(IEnumerator routine)`, mora vraćati `IEnumerator` i sadržavati barem jednu `yield` naredbu. Neke moguće `yield` naredbe su:

- `yield return null;` – izvođenje se nastavlja nakon što se obavi sljedeće iscrtavanje
- `yield return new WaitForSeconds(1.0f);` – izvođenje se nastavlja nakon što prođe jedna sekunda
- `yield return StartCoroutine(„CoroutineName");` – izvođenje se nastavlja nakon što završi izvođenje neke druge korutine

`GameLogic` sadrži sljedeće korutine:

- `StartGame()` se poziva nakon što svi klijenti *hostu* jave da su spremni za početak. U korutini se svim igračima dodjele početne karte (pozivom `RpcAddCard` funkcije u `Player` klasi), deaktivira se *loading screen* na svim klijentima i započne izvođenje sljedeće petlje:

```
while (true) {
      yield return StartCoroutine(StartRound());
      if (CheckWinConditions())
            break;
}
```

  Unutar petlje se opetovano zove korutina koja sadrži logiku jednog kruga igre, `StartRound()`. Petlja se prekida kad nakon završetka nekog kruga jedan od igrača ispuni uvjete za pobjedu, tj. kad igra završi.

- `StartRound()` za svakog igrača pozove korutinu `StartTurn(Player p)` i pričeka da završi s izvođenjem. Na kraju se pozove `RpcEndOfRound()` funkcija koja obavi sve što se treba napraviti na kraju kruga, poput dodjele zlatnika igračima i resetiranje koraka za kretanje.
- `StartTurn(Player p)` dodijeli igraču jednu kartu i obavijesti ga da je na potezu (pozivom `RpcStartTurn()` unutar `Player` klase). Zatim čeka dok igrač ne stisne gumb za kraj poteza.

Zaključak

Rezultat diplomskog rada je igra za dva do četiri igrača koja je namijenjena umreženom igranju. Igra je strateške naravi i igra se u potezima. Ploča po kojoj se igraču kreću je slučajno generirana tako da je igra drukčija pri svakom igranju. Igrači tijekom svog poteza mogu pomicati svoju figuru po poljima ploče i tako aktivirati efekte tih polja. Također mogu aktivirati efekte raznih karata koje su im dodijeljene te tako dovesti sebe u bolju poziciju, a protivnike u lošiju. Cilj igre je biti prvi igrač koji će skupiti određen broj zlatnika.

Iako su svi zahtjevi ispunjeni, uvijek ima mjesta za napredak. Postojeće mehanike mogu se poboljšati dodavanjem novih vrsta polja i karata. Moglo bi se dodati i nove mehanike koje bi igru učinile zanimljivijom te umjetnu inteligenciju u obliku igrača kojim upravlja računalo.

Literatura

[1] UNITY TECHNOLOGIES, *Multiplayer and Networking*, https://docs.unity3d.com/Manual/UNet.html

[2] UNITY TECHNOLOGIES, *Scripting API*, https://docs.unity3d.com/ScriptReference/index.html

[3] UNITY TECHNOLOGIES, *Networking Source*, https://bitbucket.org/Unity-Technologies/networking/src

[4] WIKIPEDIA, *Fisher-Yates Shuffle*, https://en.wikipedia.org/wiki/Fisher%E2%80%93Yates_shuffle

3D računalna igra inspirirana društvenim igrama

Sažetak

Rad se bavi izradom računalne igre s mehanikama koje su inspirirane društvenim igrama. Za razvoj je korišten pokretački sustav Unity, a rezultat je 3D strateška igra za dva do četiri igrača s podrškom za umreženo igranje. U sklopu rada opisana su osmišljena pravila i mehanike te su objašnjene implementacije igraćih karata, špila i slučajno generirane ploče u igri. Također je objašnjeno kako je umreženo igranje ostvareno pomoću Unityjevog sustava UNET.

Ključne riječi: razvoj igara, društvena igra, umrežavanje, Unity

3D Computer Game Inspired by Board Games

Summary

The focus of this thesis is the development of a digital board game using Unity, a popular game engine. The result is a 3D strategy game for two to four players with support for networked play. The game's rules and mechanics are described, along with the implementations of playing cards, a card deck and a randomly generated board. The thesis also contains an explanation of Unity's UNET system and how it was used to add a lobby and networking features to the game.

Keywords: game development, board game, networking, Unity

Skraćenice

UNET	*Unity Networking*	Unityjev sustav za umrežavanje
API	*Application Programming Interface*	aplikacijsko programsko sučelje
HLAPI	*High Level API*	API visoke razine
LLAPI	*Low Level API*	API niske razine
RNG	*Random Number Generator*	generator slučajnih brojeva
UI	*User Interface*	korisničko sučelje

Privitak

Instalacija programske podrške

Izvršna datoteka **game.exe** mora biti u istoj lokaciji kao i direktorij **game_Data**. Nikakva instalacija nije potrebna te se igra započinje pokretanjem izvršne datoteke.

Upute za korištenje programske podrške

Svaki korisnik koji želi sudjelovati u igri treba pokrenuti svoju kopiju igre. Jedan od igrača treba kliknuti na **Host** gumb. Ostali trebaju kliknuti na **Join**, no prije toga trebaju upisati ispravnu IP adresu (*port* je uvijek 7777).

Instance igre koje se nalaze na istom računalu kao i *host* instanca upisuju **127.0.0.1**. Inače treba saznati i upisati IP adresu *host* računala. Ako računala nisu povezana lokalnom mrežom preporučuje se korištenje virtualne privatne mreže poput LogMeIn Hamachi.

Nakon što su se svi korisnici uspješno povezali, nalazit će se u *lobbyju* u kojem mogu promijeniti svoju boju i ime. Igra će započeti kad svi korisnici klikom na gumb označe da su spremni.

Nakon što igra počne, novi se korisnici neće moći spojiti i ubaciti u igru. Također, igra će se prekinuti ako se jedan od korisnika odspoji.