Santorini – domaći zadatak

IZVEŠTAJ U URAĐENOM DOMAĆEM ZADATKU

Marija Kostić 2015/0096 | Inteligentni sistemi | Januar 2019

1. Sadržaj

2. Uvod

Domaći zadatak obuhvata implementaciju društvene igre *Santorini* i izradu izveštaja koji opisuje kako je problem rešen. Izveštaj je koncipiran tako što je prvo opisana struktura projekta. Nakon toga slede posebna poglavlja za svaki od *C#* fajlova koji su kreirani koja opisuju klase i metode iz odgovarajućeg fajla.

Zadatak je rešavan u programskom jeziku *C#* uz korišćenje *Unity* platforme.

3. Struktura rešenja

Svi fajlovi od interesa se nalaze u folderu *Assets*. U okviru njega postoji više podfoldera:

- *Art* – sadrži sve fajlove koje koristi *Unity* platforma koji određuju izgled različitih delova korisničkog interfejsa.
    - **Fantasy Skybox FREE**: folder u koje se nalaze potrebni fajlovi koji omogućavaju izgled pozadine igre.
    - **Materials**: folder koji sadrži *Unity* materijale: **HighlightMaterial.mat**, **PlayerMaterial.mat**, **RoofMaterial.mat**, **TileMaterial.mat**.
- *Prefabs* – sadrži prekonfigurisane *Unity game* objekte:
    - **Block.prefab**: objekat bloka koji igrači mogu da kreiraju.
    - **Highlight.prefab**: objekat koji pokazuje igraču koji su mu dozvoljeni potezi.
    - **Player0.prefab**: objekat koji predstavlja graditelje prvog igrača.
    - **Player1.prefab**: objekat koji predstavlja graditelje drugog igrača.
    - **Roof.prefab**: objekat krova koji igrači mogu da kreiraju.
    - **Tile.prefab**: objekat pločice koji sačinjavaju tablu i po kojima igrači mogu da se kreću i da grade.
- *Scenes* – sadrži dve *Unity* scene: **board.unity** koja sadrži sve komponente potrebne za igranje igre i **menu.unity** koja sadrži početni meni igre.
- *Scripts* – folder od najvećeg interesa. Sadrži sve *C#* skripte. Folder sadrži različite podfoldere koji reflektuju i podelu klasa i interfejsa na prostore imena. Glavni prostor imena se naziva **etf.santorini.km150096d**. Unutar njega postoje četiri prostora imena:
    - **menu**: sadrži samo jedan fajl koji je vezan za prvu (**menu.unity**) scenu.
    - **model**: sadrži klase i interfejse objekta koji modeliraju sve delove igre. U ovom prostoru imena postoje tri prostora imena: **game_objects** koji sadrži sve klase čiji su objekti vezani za odgovarajuće *Unity prefabs*, **plain_objects** koji sadrži klase čiji objekti takođe modeliraju delove igre ali nisu vezani za objekte koje korisnik može da vidi, i **interfaces** koji sadrži odgovarajuće interfejse za objekte iz prostora imena **game_objects** i **plain_objects**.
    - **moves**: sadrži klase koje modeliraju različite poteze koji igrač može da povuče, bilo da je u pitanju računar, igrač ili fajl koji sadrži poteze.
    - **utils**: sadrži *utility* klase koje se koriste za rad sa fajlovima i za pomeranje *Unity* objekata.
- *Standard Assets* – sadrži sve potrebne fajlove za koji omogućavaju korišćenje vode kao pozadine igre.

4. Korisnički zahtevi

Korisnički zahtevi se dele u sledeće grupe:

- Izgled korisničkog interfejsa
- Pokretanje igre
- Režimi rada
- Evidencija odigranih poteza
- Implementacija igrača računara

Svaka od grupa biće opisana u zasebnom podpoglavlju.

4.1. IZGLED KORISNIČKOG INTERFEJSA

U svakom trenutku treba da bude naznačeno koji igrač je na potezu i prikazana tabla za igru sa jasnim informacijama o stanju pojedinačnih polja i pozicija figura.

4.2. POKRETANJE IGRE

Pre početka igre neophodno je:

- Odabrati ko igra igru (čovek protiv čoveka, čove protiv računara ili računar protiv računara
- Omogućiti čitanje stanja igre iz fajla u vidu niza poteza koji se odigravaju do posmatranog trenutka.
- Odabrati težinu igre – dubinu razvijanja stabla igre u slučaju da je barem jedan od igrača računar.

4.3. REŽIMI RADA

Za igru koju igra računar protiv računara omogućiti dva režima rada:

- Korak po korak, gde korisnik može da isprati kompletan rad algoritma uz prikazane vrednosti funkcije procene za relevantna polja.
- Izvršavanje implementirane strategije do kraja.

4.4. EVIDENCIJA ODIGRANIH POTEZA

Bez obzira na odabrani režim rada, rešenje dodatno zapisati u vidu izlaznog fajla sa evidencijom o nizu odigranih poteza. Izlani fajl treba da ima isti format kao ulazni fajl iz koga se čita stanje igre.

4.5. IMPLEMENTACIJA IGRAČA RAČUNARA

Igrač računar treba da koristi *Minimax* algoritam. Potrebno je implementirati jednostavnog igrača koji koristi jednostavnu funkciju procene *f*. Ona se računa kao $f + m + l$, gde je m broj pločica odredišnog polja, a l broj nivoa na koje se dodaje pločica pomnožen razlikom rastojanja sopstvenih i protivničkih igrača od tog polja.

Nakon kreiranja jednostavnog igrača potrebno je implementirati jednostavnog alfa-beta igrača koji koristi alfa-beta odsecanje.

Nakon kreiranja jednostavnog alfa-beta igrača potrebno je napraviti takmičarskog alfa-beta igrača koji treba da optimizuje performanse prethodnog.

5. Opis *C#* skripti

Sve skripte se nalaze u folderu *Scripts*. Skripte će biti opisivane po prostorima imena u kojima se nalaze.

5.1. ETF.SANTORINI.KM150096D.MENU

Ovaj prostor imena je odgovoran za funkcionisanje početnog menija igre. Sadrži samo jedan fajl:

- *Menu.cs*

5.1.1. Menu.cs

Klase `Menu` sadrži reference na sve *Unity UI* objekte koji se nalaze na početnom meniju. Unutar ove klase se čuvaju svi parametri koje je korisnik odabra, na osnovu kojih se odvija igra.

```
namespace etf.santorini.km150096d.menu
{

    public class Menu : MonoBehaviour
    {
        public static Menu Instance { }
        private Menu() { }
```

Ova klasa implementira projektni uzorak *Unikat*. To implementiraju prethodno navedeno svojstvo i privatni konstruktor

```
        private void Start();
```

Metoda koja se poziva samo jednom i to prilikom stvaranja objekta. Unutar nje se poziva metoda `Init()`

```
        private void Update();
```

Metoda koja se poziva uvek prilikom osvežavanje slike. Unutar nje se proverava da li je korisnik pritisnuo dugme *Esc* koje omogućava završetak programa

```
        private void Init();
```

Unutar ove metode se radi inicijalizacija objekta. To podrazumeva postavljanje odgovarajući osluškivača na *Unity UI* objekte i postavljanje njihovog početnog stanja

```
        private void ChangePlayer(int value);
```

Metoda koja proverava da li je korisnik odabrao da barem jedan od igrača bude računar. Ima uticaja na to da li može da se aktivira režim *simulacija*

```
        private void LoadFile();
```

Metoda koja se poziva onda kada korisnik želi da učita početno stanje iz fajla.

```
        private void StartGame();
```

Poziv ove metode označava početak igre. Unutar nje se samo učitava scena **board.unity**

```
    }
}
```

5.2. ETF.SANTORINI.KM150096D.MODEL

Unutar ovog prostora imena nalaze se klase i interfejsi koji modeluju objekte igre. Sve takva klase i interfejsi su podeljeni u sledeće prostore imena:

- **game_objects**
- **plain_objects**
- **interfaces**

5.2.1.etf.santorini.km150096d.model.interfaces

Ovaj prostor imena sadrži interfejse za odgovarajuće objekte iz prostora imena **game_objects** i **plain_objects**. Sadrži sledeće fajlove:

- *IBoard.cs*
- *IPlayer.cs*
- *ITile.cs*

5.2.1.1. IBoard.cs

Fajl sadrži istoimeni interfejs, koji opisuje tablu igre. Svaka tabla ima objekte svojih pločica koji mogu da se dohvate. Takođe ima i objekte svojih igrača koji takođe mogu da se dohvate. Pored toga svaka tabla čuva informaciju o tome koji igrač je trenutno na potezu i o parametrima igre koje je postavio korisnik.

```
namespace etf.santorini.km150096d.model.interfaces
{
    public interface IBoard
    {
        ITile this[int x, int y] { get; }
        ITile this[Vector2 position] { get; }
```

Svojstva za dohvatanje pločice table i to na osnovu koordinata koje se zadaju ili direktno ili preko objekta klase `Vector2`.

```
        IPlayer this[PlayerID id, int num] { get; set; }
```

Svojstvo za dohvatanje i postavljanje igrača/graditelja na osnovu njegovog tipa i rednog broja zbog toga što svaki igrač ima dva graditelja.

```
        IPlayer SelectedPlayer { get; set; }
```

Svojstvo za dohvatanje i postavljanje trenutno selektovanog igrača/graditelja.

```
        PlayerID TurnId { get; set; }
```

Svojstvo za dohvatanje i postavljanje tipa igrača koji je trenutno na potezu .

```
        bool Simulation { set; get; }
        int MaxDepth { set; get; }
```

Svojstva za dohvatanje i postavljanje parametara igre koji označavaju da li je trenutno aktivan režim *simulacije* i maksimalnu dubinu stabla igre

```
        void ChangeTurn();
```

Metoda koja menja igrača koji je trenutno na potezu.

```
    }
}
```

5.2.1.2. IPlayer.cs

U ovom fajlu se nalaze nabrojivi tip `PlayerID` koji određuje igrača, da li je u pitanju prvi ili drugi igrač i interfejs `IPlayer` koji opisuje jednog igrača/graditelja.

```
namespace etf.santorini.km150096d.model.interfaces
{
    public enum PlayerID : int { PLAYER0 = 0, PLAYER1 };

    public interface IPlayer
    {
        PlayerID Id { get; set; }
        Vector2 Position { get; set; }
```

Svojstva za dohvatanje i postavljanje informacija o svakom igraču/graditelju, a to su njegov id i njegova pozicija na tabli.

```
    }
}
```

5.2.1.3. ITile.cs

Fajl sadrži istoimeni interfejs koji opisuje jednu pločicu na tabli. Pored toga sadrži i nabrojivi tip `Height` koji određuje broj izgrađenih blokova na pločici.

```
namespace etf.santorini.km150096d.model.interfaces
{
    public enum Height { H0 = 1, H1, H2, H3, ROOF };
    public interface ITile
    {
        Height Height { get; set; }
```

Svojstvo za dohvatanje i postavljanje visine blokova koji su izgrađeni na pločici.

```
        IPlayer Player { get; set; }
        bool HasPlayer();
```

Na pločici može da se nalazi igrač/graditelj. Informacije o tome da li se igrač/graditelj nalazi na pločici i mogućnost njegovog postavljanja i dohvatanja omogućavaju prethodno navedeno svojstvo i metoda.

```
    }
}
```

5.2.2. etf.santorini.km150096d.model.game_objects

Ovaj prostor imena sadrži klase svih objekata koji se mogu naći na tabli i vezani su za odgovarajuće *Unity prefabs.* Sadrži sledeće fajlove:

- *Block.cs*
- *Board.cs*
- *Highlight.cs*
- *Player.cs*
- Roof.cs
- Tile.cs

5.2.2.1. Block.cs

Fajl sadrži istoimenu klasu koja opisuje objekte blokova koje igrači mogu da izgrade. Ima samo jednu statičku metodu.

```
namespace etf.santorini.km150096d.model.gameobject
{
    public class Block : MonoBehaviour
    {
        public static void GenerateBlock(int x, int y, Board board, int height);
```

Statička metoda koja kreira jedan objekat ovog tipa i postavlja ga na odgovarajuće mesto na zadatoj tabli.

```
    }

}
```

5.2.2.2. Board.cs

Fajl sadrži klasu `Board` koja je centralna klasa projekta. U njoj se nalaze svi *Unity game* objekti na osnovi kojih se generišu objekti na tabli. Takođe, ova klasa čuva i reference na *Unity UI* objekte preko kojih se korisniku prikazuju informacije o statusu igre i o radu algoritma u slučaju da je barem jedan od igrača računara a igra je pokrenuta u režimu *simulacije*. Ova klasa takođe sadrži i objekte tipa `Move` koji su zaduženi za realizaciju poteza.

U prikazu metoda ove klase izostavljene su metode interfejsa `IBoard` koji klasa implementira.

```
namespace etf.santorini.km150096d.model.gameobject
{
    public class Board : MonoBehaviour, IBoard
    {
        public static IBoard Instance { get; set; }
```

```
private Board() { }
```

Klasa implementira projektni uzorak *Unikat*. O omogućavaju prethodno navedeno svojstvo i privatni konstruktor.

```
private void Start();
```

Ova metoda se poziva samo jednom i to prilikom kreiranja objekta ove klase. Ona služi za inicijalizovanje sistema. To podrazumeva generisanje same table, preuzimanje parametara igre iz klase `Menu`, kreiranje odgovarajućih objekata klasa izvedenih iz `Menu` i postavljanje početnog sadržaja *Unity UI* komponenti.

```
private void Update();
```

Ova metoda se poziva prilikom svakog osvežavanja slike (na svaki frejm). Unutar nje se proverava da li je korisnik želi da ugasi program, ažurira se trenutna pozicija miša, zatim se odigrava potez i na kraju proverava da li je igra završena.

```
private void OnApplicationQuit();
```

Metoda koja se poziva kada treba ugasiti program. Unutar nje se poziva metoda za čuvanje izlaznog fajla.

```
private void UpdateMouseOver()
private void ResetMouseOver()
private bool MouseInsideBoard()
```

Tri metode koje rade sa trenutnom pozicijom miša. Prva određuje trenutnu poziciju miša, druga je resetuje, a treća proverava da li je pozicija miša unutar tabele.

```
private void GenerateBoard();
```

Metoda koja generiše tablu i sve njene pločice.

```
private void CheckGameOver();
```

Metoda koja se poziva is metode `Update()`. Proverava da li je igra završena i ažurira odgovarajući *Unity UI* objekat.

```
public bool IsGameOver(ref PlayerID winnerId);
```

Metoda koja proverava da li je igra završena i ako jeste kroz parametar winnerId vraća id pobednika.

```
public void MakeMove(Vector2 position);
public bool HasPossibleMoves();
public bool MouseInputNeeded();
```

Ove tri metode pozivaju istoimene metode objekta klase `Move`, i to za onog igrača koji je trenutno na redu.

```
public void UpdateMessage(string message);
```

```
        public void AddToSimulationLog(string message);
        public void ResetSimulationLog();
```

Ove tri metode su zadužene za postavljanje zadatog teksta na *Unity UI* objekte.

```
        public void InitMenuInfo();
```

Metoda koja čita parametre igre iz objekta klase `Menu`. Poziva se jednom i to prilikom inicijalizacije sistema u metodi `Start()`.

```
        public void InitPlayerMoves();
```

Poziva se isto kad i prethodna metoda. Na osnov parametara igre kreira odgovarajuće objekte klase izvede iz klase `Move`.

```
        public void StartReadingFromFile();
        public void FinishReadingFromFile();
```

Ove dve metode omogućavaju pravila rad sistema kada korisnik želi da početno stanje igre učita iz zadatog fajla.

```
        public void SetHighlight(bool[,] possibleMoves);
        public void ResetHighlight();
```

Ove dve metode su zadužene za prikazivanje odnosno uklanjanje mogućih poteza igrača sa table.

```
    }
}
```

5.2.2.3. Highlight.cs

Fajl sadrži istoimenu klasu koja opisuje objekte koji igračima pokazuju sve poteze koje mogu da povuku. Ima samo jednu statičku metodu.

```
namespace etf.santorini.km150096d.model.gameobject
{
    public class Highlight : MonoBehaviour
    {
        public static Highlight GenerateHighlight(int x, int y, Board board);
```

Statička metoda koja kreira jedan objekat ovog tipa.

```
   }

}
```

5.2.2.4. Player.cs

Fajl sadrži istoimenu klasu koja je vezana za odgovarajući *Unity* objekat graditelja.

U prikazu metoda ove klase izostavljene su metode interfejsa `IPlayer` koji klasa implementira.

```
namespace etf.santorini.km150096d.model.gameobject
{
    public class Player : MonoBehaviour, IPlayer
    {
        public static IPlayer GeneratePlayer(PlayerID id, Vector2 position, Board
board);
```

Statička metoda koja kreira jedan objekat ovog tipa i postavlja ga na odgovarajuće mesto na tabli

```
    }
}
```

5.2.2.5. Roof.cs

Fajl sadrži istoimenu klasu koja opisuje objekat krova koji igrači/graditelji mogu da naprave. Ima samo jednu statičku metodu.

```
namespace etf.santorini.km150096d.model.gameobject
{
    public class Roof : MonoBehaviour
    {
        public static void GenerateRoof(int x, int y, Board board , int height);
```

Statička metoda koja kreira jedan objekat ovog tipa i postavlja ga na odgovarajuće mesto na tabli.

```
   }
}
```

5.2.2.6. Tile.cs

Fajl sadrži istoimenu klasu koja je vezana za odgovarajući *Unity* objekat pločice.

U prikazu metoda ove klase izostavljene su metode interfejsa `ITile` koji klasa implementira.

```
namespace etf.santorini.km150096d.model.gameobject
{
    public class Tile : MonoBehaviour, ITile
    {

        private void Start();
```

Ova metoda se poziva samo jednom, i to prilikom kreiranja objekta. Koristi se za inicijalizaciju objekta.

```
        public static Tile GenerateTile(int x, int y, Board board);
```

Statička metoda koja kreira jedan objekat ovog tipa i postavlja ga na odgovarajuće mesto na tabli.

```
        public void SetHighlight(int x, int y);
        public void ResetHighlght();
```

Svaka pločica koja je vezana za *Unity* objekat ima i svoj objekat tipa `Highlight`. Ove dve metode se koriste da prikažu odnosno sakriju ovaj objekat.

```
    }
}
```

5.2.3. etf.santorini.km150096d.model.plain_objects

Ovaj prostor ima sadrži obične (*PO – Plain Object*) objekte koji implemetiraju interfejse is prostora imena **interfaces**. Objekte ovih klasa koriste objekti odgovarajući klasa izvedenih iz `Move` koje izvršavaju algoritme za igrače računare. Zbog rekurzivnih poziva tih algoritama bile je potrebno kreirati kopije table u različitim stanjima što nije moguće kada su u pitanju *Unity* objekti. Ovaj prostor ima sadrži sledeće fajlove:

- *BoardPO.cs*
- *PlayerPO.cs*
- TilePO.cs

5.2.3.1. BoardPO.cs

Ovaj fajl sadrži istoimenu klasu koja predstavlja tablu koja nije vezana za *Unity* objekte.

U prikazu metoda ove klase izostavljene su metode interfejsa `IBoard` koji klasa implementira.

```
namespace etf.santorini.km150096d.model.plain_objects
{
    public class BoardPO : IBoard
    {
        public BoardPO(IBoard board);
```

Konstruktor ove klase kreira novi objekat kao kopiju zadatog objekta

```
        private void CopyTiles(IBoard board);
        private void CopyPlayers(IBoard board);
```

Prethodne dve metode su zadužene da kopiraju odgovarajuće podatke iz zadate table. Potrebne su zato što se svi klasni objekti i nizovi u *C#* prenose po referenci. Ovde se pravi „duboka“ kopija.

```
    }
}
```

5.2.3.2. PlayerPO.cs

Ovaj fajl sadrži istoimenu klasu koja predstavlja jednog igrača/graditelja koji niji vezana za *Unity* objekat.

U prikazu metoda ove klase izostavljene su metode interfejsa `IPlayer` koji klasa implementira.

```
namespace etf.santorini.km150096d.model.plain_objects
{
    public class PlayerPO : IPlayer
    {
```

Nema drugih metoda pored onih u interfejsu `IPlayer.`

```
    }

}
```

5.2.3.3. TilePO.cs

Ovaj fajl sadrži istoimenu klasu koja predstavlja pločicu koja nije vezana za *Unity* objekte.

U prikazu metoda ove klase izostavljene su metode interfejsa `ITile` koji klasa implementira.

```
namespace etf.santorini.km150096d.model.plain_objects
{
    public class TilePO : ITile
    {
```

Nema drugih metoda pored onih u interfejsu `ITile.`

```
    }

}
```

5.3. ETF.SANTORINI.KM150096D.MOVES

Ovaj prostor imena sadrži sve klase koje predstavljaju različite načine na koji potez može da se povuče. Ove klase u stvari apstrahuju povlačenje jednog poteza. Prostor imena sadrži sledeće fajlove:

- *Move.cs*
- *HumanMove.cs*
- *FileMove.cs*
- *AIMove.cs*
- *EasyMove.cs*
- *MediumMove.cs*
- *HardMove.cs*

5.4. Move.cs

Ovaj fajl sadrži nabrojivi tip `MoveType` koji opisuje sve moguće vrste načina na koji se potez može povući.

U ovom fajlu se takođe nalazi apstraktna klasa `Move` koja apstrahuje povlačenje jednog poteza. Sve ostale klase iz ovog prostora imena se izvode iz nje. Svaki objekat ove klase ima svoju tablu nad kojom manevriše i tip igrača na koji e odnosi. Povlačenje poteza je koncipirano tako da se prolazi kroz automat stanje. Zbog toga je uveden još jedan nabrojivi tip `MoveState` koji opisuje sva stanja u kojima se igrač može naći dok povlači poteze. Ovo je potrebno zbog specifičnosti igre da se jedan potez u stvari sastoji od više manjih poteza (selektovanje graditelja, njegovo pomeranje i na kraju izgradnja boka). Ova klasa implementira sve metode koje manipulišu sa tablom. Jedino što klase izvedene iz ove treba da menjaju jeste način dobijanja pozicije poteza. To može biti iz fajla, preko klika korisnika igre ili kao rezultat algoritma.

```
namespace etf.santorini.km150096d.moves
{
    public abstract class Move
    {
        public Move(PlayerID id, IBoard board)
        public void CopyMove(Move move);
        public void CopyPossibleMoves(Move move);
```

Ove tri metode služe sa konstrukciju objekta ove klase. Prvo je konstruktor a naredne dve metode se koriste za duboko kopiranje odgovarajućih polja objekta.

```
        public abstract bool MouseInputNeeded();
```

Apstraktna metoda koja treba da vrati informaciju o tome da li je potrebno da korisnik napravi klik mišem radi povlačenja poteza.

```
        public virtual void MakeMove(Vector2 position);
```

Ovo je glavna metoda ove klase. Unutar nje se na osnovu trenutnog stanja poteza pozivaju odgovarajuće metode i menja stanje poteza. Klase koje se izvode iz ove klase ovu metodu menjaju tako što pre nego što pozovu metodu osnovne klase na neki način dobiju poziciju poteza.

```
        protected bool SelectPlayer(Vector2 position);
        protected bool MovePlayer(Vector2 dstPosition);
        protected bool Build(Vector2 position);
```

Ove tri metode implementiraju odgovarajući potez igrača na osnovu trenutnog stanja poteza. Prva selektuje graditelja, druga ga pomera, a treća izgrađuje novi blok/krov.

```
        protected void CalculatePossibleMoves(Vector2 playerPosition);
        protected void CalculatePossibleBuilds(Vector2 playerPosition);
```

Ove dve metode se koriste da izračunaju naredni mogući potez igrača. Prva izračunava gde selektovani graditelj može da se pomeri a druga gde može da izgradi blok/krov.

```
public bool IsWinner();
```

Metoda vraća informaciju o tome da li je igrač pobedio tako što se popeo na pločicu sa izgrađena tri bloka.

```
protected bool PositioningIsPossible(Vector2 position);
```

Vraća informaciju o tome da li na prosleđeno mesto može da se postavi graditelj na početku igre.

```
public bool HasPossibleMoves();
public bool CanMove(Vector2 playerPosition);
public bool CanBuild(Vector2 builderPosition);
```

Ove tri metode određuju da li postoji potez koji igrač može da povuče. Prva metoda poziva jednu od naredne dve u zavisnosti od toga u kom stanje se potez trenutno nalazi.

```
public static Move CreateMove(MoveType type, PlayerID id, IBoard board);
```

Statička metoda koja kreira objekat odgovarajuće klase izvede iz `Move` na osnovu prosleđenog parametra `type`.

U svim narednim prikazima klasa izvedenih iz `Move` biće prikazane metode koje su implementacija apstraktnih metoda osnovne klase, kao i metode koje su redefinisane. Takođe će biti prikazane i nove metode kojih nema u osnovnoj klasi.

5.5. HumanMove.cs

Ovaj fajl sadrži klasu koja opisuje potez koji treba da povuče korisnik.

```
namespace etf.santorini.km150096d.moves
{
    public class HumanMove : Move
    {
        public override bool MouseInputNeeded();
```

Ovoj klasi je kao ulaz za povlačenje poteza potrebna pozicija koja se dobija na osnovu korisnikovog klika miša.

```
    }

}
```

5.6. FileMove.cs

Ovaj fajl sadrži klasu koja opisuje potez koji se povlači na osnovu sadržaja ulaznog fajla.

```
namespace etf.santorini.km150096d.moves
{
    class FileMove : Move
      {

          public override bool MouseInputNeeded();
```

Klik miša nije potreban, naredna pozicija se čita iz fajla.

```
        public override void MakeMove(Vector2 position);
```

Pre poziva istoimene metode osnovne klase pozicija narednog poteza se dobija čitanjem iz ulaznog fajla.

```
    }
}
```

5.7. *AIMove.cs*

Ovaj fajl sadrži klasu koja apstrahuje poteze koje povlači računar na osnovu algoritma.

```
namespace etf.santorini.km150096d.moves
{
    public abstract class AIMove : Move
    {
        public sealed override bool MouseInputNeeded();
```

Klik miša je potreban jedino kada je igra pokrenuta u režimu *simulacije*. To znači da je potrebno da korisnik klikne bilo gde da bi računar povukao svoj potez. Neme potrebe da klase izvedene iz ove preklapaju ovu metodu.

```
        public sealed override void MakeMove(Vector2 position);
```

Bez obzira na način računanja narednog poteza računar prilikom povlačenja poteza treba da ga izračuna i ako je potrebno da ga ispiše korisniku preko nekog od *Unity UI* objekata. Zbog toga je ove metoda `sealed`.

```
        protected abstract float Algorithm(Vector2[] bestMove, int currentDepth,
float alpha, float beta);
```

Metoda koju izvede klase moraju da implementiraju. Ona vraća najbolji potez na osnovu algoritma. Predviđeno je da se zove rekurzivno.

```
        protected abstract float Evaluate(Vector2[] bestMove);
```

Statička funkcija procene koju koristi algoritam. Parametar je potez koji je doveo do trenutnog stanja table.

```
        private Vector2 RandomPosition();
```

Vraća nasumičnu poziciju na tabli. Koristi se za pozicioniranje graditelja na početku igre.

```
        protected AIMove CopyBoardAndCreateMove(Move other, MoveType type);
```

Metoda koja kreira novi objekat odgovarajuće klase izvedene iz `Move` na osnovu prosleđenog tipa. Takođe pravi i kopiju prosleđene table. Koristi se kao priprema za rekurzivni poziv algoritma.

```
        private void CopyWinningPlayer(AIMove other);
```

Metoda koja kopira podatke o igraču/graditelju koji je najbliži pobedi.

```
        protected List<AIMove> GetAllPossibleMoves();
```

Meto koja vraća listu svih poteza koji igrač može da napravi.

```
    }
}
```

5.8. EasyMove.cs

Ovaj fajl sadrži istoimenu klasu koja implementira jednostavnog igrača.

```
namespace etf.santorini.km150096d.moves
{
    public class EasyMove : AIMove
    {
        // MINIMAX
        protected override float Algorithm(Vector2[] bestMove, int currentDepth,
float alpha, float beta);
```

Algoritam koji implementira ova vrsta poteza je *Minimax*. Povratna vrednost je funkcija procene za povučeni potez. Preko parametra `bestMove` se vraća najbolji potez. Algoritam radi tako što prvo proverava da li je stanje table terminalno. U slučaju da jeste vraća se odgovarajuća vrednost. Zatim se proverava da li se stiglo do najdubljeg dozvoljenog rekurzivnog poziva. Ako jeste vraća se statička funkcija procene. U slučaju da nije pronalaze se svi mogući potezi trenutnog igrača. Za svakog od dva graditelja traže se sve moguće pozicije na koje oni mogu da se pomere. Zatim se za sve te nove pozicije traže sve moguće pozicije na koje izabrani graditelj može da sagradi blok/krov. Za sve takve pozicije se poziva ista ova funkcija samo ne nad istim objektom. Sva ova „probna“ stanja table su u stvari kopije originalnog stanja table sa povučenim jednim potezom.

```
        protected override float Evaluate(Vector2[] move);
```

Metoda koja računa zadatu statičku funkciju procene.

```
        private float PlayerDistance(Vector2 position);
```

Pomoćna metoda koja računa rastojanja koja su potrebna prethodnoj metodi.

```
    }

}
```

5.9. MediumMove.cs

Ovaj fajl sadrži istoimenu klasu koja implementira jednostavnog igrača sa alfa-beta odsecanjem.

```
namespace etf.santorini.km150096d.moves
{
    public class MediumMove : AIMove
    {
        // ALPHA BETA MINIMAX
        protected override float Algorithm(Vector2[] bestMove, int currentDepth,
float alpha, float beta);
```

Algoritam koji implementira ova vrsta poteza je *Minimax* sa alfa-beta odsecanjem. Funkcioniše isto kao i kod klase `EasyMove` s tom razlikom da uzima u obzir parametre `alpha` i `beta`. To može dovesti do manjeg broja rekurzivnih poziva.

```
        protected override float Evaluate(Vector2[] move);
```

Metoda koja računa zadatu statičku funkciju procene. Ista kao kod `EasyMove`.

```
        private float PlayerDistance(Vector2 position);
```

Pomoćna metoda koja računa rastojanja koja su potrebna prethodnoj metodi. Ista kao kod `EasyMove`.

```
    }

}
```

5.10. *HardMove.cs*

Ovaj fajl sadrži istoimenu klasu koja implementira naprednog igrača sa alfa-beta odsecanjem.

```
namespace etf.santorini.km150096d.moves
{
    public class HardMove : AIMove
    {
        // ALPHA BETA MINIMAX
        protected override float Algorithm(Vector2[] bestMove, int currentDepth,
PlayerID player, float alpha, float beta);
```

Algoritam koji implementira ova vrsta poteza je *Minimax* sa alfa-beta odsecanjem. Ista kao kod `MediumMove`.

```
        protected override float Evaluate(Vector2[] move);
```

Metoda koja računa statičku funkciju procene. Po ovome se algoritam razlikuje od onog u klasi `MediumMove`. U svakoj situaciji na tabli postoji graditelj/igrač koji je najbliži pobedi. Ako je to isti igrač koji pravi potez funkcija procene će biti bolja za one poteze koji podrazumevaju pomeranje dalje od protivnika i građenje na istom ili jednom više nivou. Ako je najbliži pobedi protivnički igrač, najbolji potez treba da povuče graditelj tako što će da se pomeri ka njemu i gradi tako da mu smanji mogući broj poteza. Funkcija procene je zbir ove dve vrednosti.

```
       private float DistanceFromOpponent(Vector2 position, PlayerID opponentId);
```

Metoda vraća zbir rastojanja od zadate pozicije do pozicija graditelja zadatog igrača.

```
        protected override void FindWinningPlayer();
```

Metoda određuje koji je od ukupno četiri graditelja najbliži pobedi po određenoj heuristici.

```
        private int WinValue(IPlayer player);
```

Za zadatog igrača metoda računa heurističku funkciju koja se koristi u prethodno navedenoj metodi. Povratna vrednost je veća što zadati igrač ima više mesta na koja može da se pomeri. Mesta sa većom visinom doprinose više.

```
    }

}
```

5.11. ETF.SANTORINI.KM150096D.UTILS

U ovom prostoru imena se nalaze dve uslužne klase i to u dva fajla:

- *Util.cs*
- *FileManager.cs*

5.11.1. Util.cs

Ovaj fajl sadrži istoimenu klasu. Sve metode osim jedne su statičke i imaju ulogu da prosleđene *Unity* objekte postave na odgovarajuće mesto na tabli.

```
namespace etf.santorini.km150096d.utils
{
    public class Util
    {
        public static void MoveTile(Tile tile, int x, int y, int height);
        public static void MoveHighlight(Highlight highlight, int x, int y, int
height);
        public static void MovePlayer(Player player, int x, int y, int height);
        public static void MoveBlock(Block block, int x, int y, int height);
        public static void MoveRoof(Roof roof, int x, int y, int height);
```

Prosleđene objekte postavljaju na odgovarajuće mesto na tabli.

```
        public static float Distance(Vector2 first, Vector2 second);
```

Ova metoda vraća rastojanje između dva vektora.

```
    }
}
```

5.11.2. FileManager.cs

Fajl sadrži istoimenu klasa koja se koristi za rad sa fajlovima (i ulaznim i izlaznim).

```
namespace etf.santorini.km150096d.utils
{
```

```
public class FileManager
{

    private FileManager() { }
    public static FileManager Instance { get; }
```

Klasa implementira uzorak *Unikat*, a to omogućavaju privatni konstruktor i prethodno svojstvo.

```
    public Vector2 Src { get; set; }
    public Vector2 Dst { get; set; }
    public Vector2 Build { get; set; }
```

Svojstva za dohvatanje i postavljanje pozicija koje će kasnije biti upisane u izlazni fajl.

```
    public bool SetInput(string fileName);
```

Kreira ulazni tok podataka za zadati naziv fajla.

```
    public bool HasNextPosition();
```

Vraća informaciju o tome da li u fajl postoji još pozicija ili se stiglo do kraja sa čitanjem.

```
    public Vector2 NextPosition();
```

Vraća narednu poziciju iz ulaznog fajla.

```
    public void SaveFile();
```

Zatvara izlazni fajl.

```
    public void SetOutput();
```

Kreira izlazni tok podataka.

```
    public void WritePositions();
```

U fajl upisuje početne pozicije graditelja.

```
    public void WriteMove();
```

U fajl upisuje potez jednog igrača/graditelja po zadatom formatu.

```
    private void WriteChar(char c);
```

Upisuje jedan karakter fajl.

```
    private void WriteVector(Vector2 position);
```

U fajl upisuje zadatu poziciju.

```
}
}
```