SI4IS

Santorini – društvena igra

INTELIGENTNI SISTEMI

PREDRAG MITROVIĆ 2015/0608

ELEKTROTEHNIČKI FAKULTET U BEOGRADU | 2018/2019

SADRŽAJ

SPISAK IZMENA

Datum	Verzija	Mesta izmene	Autor
01.01.2019.	1.0	Osnovna verzija	Predrag Mitrović

1. Uvod

Tema projektnog zadatka je implementacija društvene igre Santorini. Akcenat projektnog zadatka je na implementaciji AI botova različite težine. Logika igrice je programirana u programskom jeziku C#, uz korišćenje Unity engine-a.

2. Implementacija

2.1. interface Player

Sadrži akcije koje igrač preduzima u toku igre.

```
/* računanje sledećeg poteza */
Task CalculateNextMove();
/* izbor početnog polja figure na početku igre */
Task PickStartingField(SemaphoreSlim semaphore);
/* izbor figure kojom će biti odigran naredni potez */
Task SelectFigure(SemaphoreSlim semaphore);
/* izbor polja na koje će biti pomerena prethodno izabrana figura */
Task MoveFigure(SemaphoreSlim semaphore);
/* izbor polja koje će biti nadograđeno nakon pomeranja figure */
Task BuildNewLevel(SemaphoreSlim semaphore);
```

2.2. class Human: Player

Predstavlja implementaciju igrača kog kontroliše korisnik.

2.3. class RandomBot: Player

Predstavlja implementaciju bota koji poteze odigrava na nasumičan način. Korišćen je za početno testiranje stabilnosti sistema.

2.4. class MinimaxBot: Player

Predstavlja implementaciju bota koji pomoću Minimax algoritma izračunava najoptimalniji potez. Detaljnije u sekciji 3.

2.5. class AlphaBetaBot: Player

Predstavlja nadogradnju MinimaxBot-a koji koristi alfa-beta odsecanje da ubrza proces izračunavanja najoptimalnijeg poteza. Detaljnije u sekciji 3.

2.6. class AdvancedBot: Player

Predstavlja nadogradnju AlphaBetaBot-a koji koristi poboljšanu funkciju procene za izračunavanje optimalnog poteza. Detaljnije u sekciji 3.

2.7. class LoadGameBot: Player

Predstavlja dummy igrača koji se koristi pri učitavanju igre. Odigrava poteze u skladu sa učitanim podacima iz fajla.

2.8. class GameController: MonoBehaviour

Skripta odgovorna za logiku igrice.

```
/* pokretanje igre */
async void StartGame();
/* završetak igre */
void FinishGame();
/* promena igrača na potezu, nakon završetka poteza */
void SwitchTurns();
/* kreiranje figure igrača sa zadatim materijalom */
void InstantiatePlayerFigure(Material material);
/* kreiranje objekata igrača (npr. new MinimaxBot()…) */
void InstantiatePlayers();
/* vizelni prikaz mogućih pomeraja igrača, vraća broj mogućnosti */
int ShowPossibleMoves();
/* broj mogućnosti pomeraja za figuru sa zadatom pozicijom */
int CountPossibleMoves(string currentPosition);
/* da li igrač na potezu nema više mogućih poteza */
bool NoMovesLeft();
/* vizuelni prikaz mogućih nadogradnja nakon pomeraja figure */
void ShowPossibleBuilds();
/* vizuelna transformacija pomeraja figure */
void MoveFigure();
/* vizuelna transformacija nadogradnje nivoa */
void BuildNewLevel();
/* omogućava igraču da klikne na dozvoljena polja */
void EnableAllFields();
/* zabranjuje igraču da klikne na dozvoljena polja */
void DisableAllFields();
/* omogućava igraču da klikne na dozvoljene figure */
void EnableAllFigures();
/* zabranjuje igraču da klikne na dozvoljene figure */
void DisableAllFigures();
/* omogućava igraču koji je na potezu da klikne na dozvoljene figure */
void ActivatePlayerFigures();
/* vraća i-ti objekat figure zadatog igrača */
GameObject FetchMyFigure(Player p, int i);
/* vraća objekat figure na zadatoj poziciji */
public GameObject FetchFigure(string currentPosition);
/* briše iz igre dummy igrača nakon što odigra sve učitane poteze */
void RemoveLoadGameBot(LoadGameBot bot);
```

2.9. class GameState

Klasa koja sadrži trenutno informacije o trenutnom stanju table za igru. Sadrži unutrašnju klasu GameField koja sadrži informacije o nivou polja i o tome da li postoji figura koja se nalazi na tom polju.

```
/* izračunava minimalnu distancu za zadatog igrača od zadatog polja */
float Distance(int player, string field)
/* izvršava zadati potez */
void MakeMove(Logger.GameMove move);
/* negira zadati potez */
void UndoMove(Logger.GameMove move);
/* menja igrača na potezu */
void ChangeTurns();
/* vraća sve moguće pomeraje figure na zadatoj poziciji */
List<Logger.GameMove> GetPossibleMoves(string currentPosition);
/* vraća sve moguće nadogradnje polja nakon zadatog pomeraja figure */
List<Logger.GameMove> GetPossibleBuilds(Logger.GameMove gameMove);
/* vraća sve moguće nadogradnje polja nakon svih zadatih pomeraja figure */
List<Logger.GameMove> GetPossibleBuilds(List<Logger.GameMove> possibleMoves);
/* vraća pozicije figure igrača koji je na potezu */
List<string> GetCurrentPlayerPositions();
```

2.10. class Logger

Klasa odgovorna za snimanje toka igre u fajl i učitavanje toka igre iz fajla. Sadrži unutrašnju klasu GameMove koja sadrži informacije o prethodnoj poziciji figure, narednoj poziciji figure i narednoj poziciji nadogradnje polja.

```
/* čuva informaciju o početnoj poziciji i o narednoj poziciji figure */
void LogStartingPosition(Player current, string firstFigurePosition, string
secondFigurePosition);
/* menja informaciju o nadogradnji polja */
void AlterLastBuildPosition(Player current, string newLevelBuildPosition);
/* čuva informaciju o celokupnom potezu */
void LogGameMove(Player current, string previousFigurePosition, string
nextFigurePosition, string newLevelBuildPosition);
/* upisuje listu poteza u fajl */
void WriteToFile();
/* učitava listu poteza iz fajla */
void ReadFromFile();
```

2.11. class PlayerFigure: MonoBehaviour

Skripta koja handle-uje event-ove korisnika sa figurama igrača.

2.12. class Field: MonoBehaviour

Skripta koja handle-uje event-ove korisnika sa poljima na tabli.

2.13. class PrefabSpawner: MonoBehaviour

Skripta koja handle-uje zahteve za instanciranje GameObject objekata.

2.14. class UserInterface: MonoBehaviour

Skripta koja je odgovorna za interakciju korisnika sa korisničkim interfejsom u igri.

2.15. class MainMenu: MonoBehaviour

Skripta koja je odgovorna za interakciju korisnika sa glavnim menijem.

2.16. class PlayMenu: MonoBehaviour

Skripta koja je odgovorna za interakciju korisnika sa menijem za početak igre.

3. AI algoritmi

```
Logger.GameMove Minimax(GameState currentState, bool maximizingPlayer, out float?
est, int maxLevel = 3, int current = 0);
Logger.GameMove AlphaBeta(GameState currentState, bool maximizingPlayer, out float?
est, float? alpha, float? beta, int maxLevel = 3, int current = 0);
float Estimate(GameState state, Logger.GameMove move);
```

MinimaxBot koristi Minimax algoritam koji se u teoriji igara koristi za minimizaciju mogućih gubitaka, odnosno maksimizaciju minimalnog dobitka. AlphaBetaBot koristi isti algoritam uz modifikaciju koja se naziva alfa-beta odsecanje koja potencijalno omogućava smanjenje broja čvorova za pretragu, te samim tim ubrzava algoritam. Pri implementaciji ova dva bota korišćena je funkcija procene (Estimate) koja je zadata u projektnom zadatku. AdvancedBot predstavlja nadogradnju AlphaBetaBot-a funkcijom procene koja bi trebalo da ima bolje performanse od prethodno korišćene.
Modifikacija se sastoji u tome da:

1. Umesto razlike minimalnih distanci između trenutnog igrača i protivnika, uzima se minimalna distanca između protivnika i trenutnog igrača. Za trenutnog igrača optimalno je da ova razlika bude što veća, odnosno da protivnikova minimalna distanca bude što veća, a suopstvena minimalna distanca što manja.
2. Pri izračunavanju prethodno navedenog parametra, uveden je specijalan slučaj kada se razmatra gradnja kupole. Prethodna statička funkcija procene je uticala na to da bot ponekad gradi kupolu iako je protivnik daleko, dakle bez potrebe. Ovde se funkcija procene povećava znatno ukoliko je protivnik na jedan korak od polja nivoa 3, u zavisnosti od toga na polju kog nivoa se nalazi protivnik. Što je na većem nivou, to znači da je bliži pobedi i da je potrebno da trenutni igrač izgradi kupolu na zadatom polju kako ne bi izgubio.
3. Dodat je parametar q u funkciju procene koji uvećava funkciju procene u zavisnosti od nivoa polja na kom se nalazi figura trenutnog igrača koja je na minimalnom rastojanju od polja koje će biti nadograđeno. Praktično, trenutnom igraču je optimalno da bude na što višem nivou polja koje je najbliže novoizgrađenom polju i da protivnik bude na što manjem nivou polja koje je najdalje novoizgrađenom polju.

Ispitivan je i uticaj različitih težinskih koeficijenata koji idu uz odgovarajuće parametre, ali je ustanovljeno da je najoptimalnije da težinski koeficijenti budu 1 uz sve parametre.