S V E U Č I L I Š T E U S P L I T U
FAKULTET ELEKTROTEHNIKE, STROJARSTVA I BRODOGRADNJE

SEMINARSKI RAD

ASSASINS

Izradili:

Luka Rančić
Stipe Žeravica
Dujam Krstulović

Split, Siječanj, 2019.

Sadržaj

1. Uvod

U ovom seminarskom radu prikazan je postupak izrade 3D desktop igre/aplikacije naziva Assasins. Assasins je trodimenzionalna igra na ploči u kojoj je cilj doći do prethodno definirane lokacije na ploči pri čemu je moguće eliminirati "protivnike" na način da se dođe do pozicije na kojoj se oni nalaze u trenutku dok nismo u njihovom fokusu.

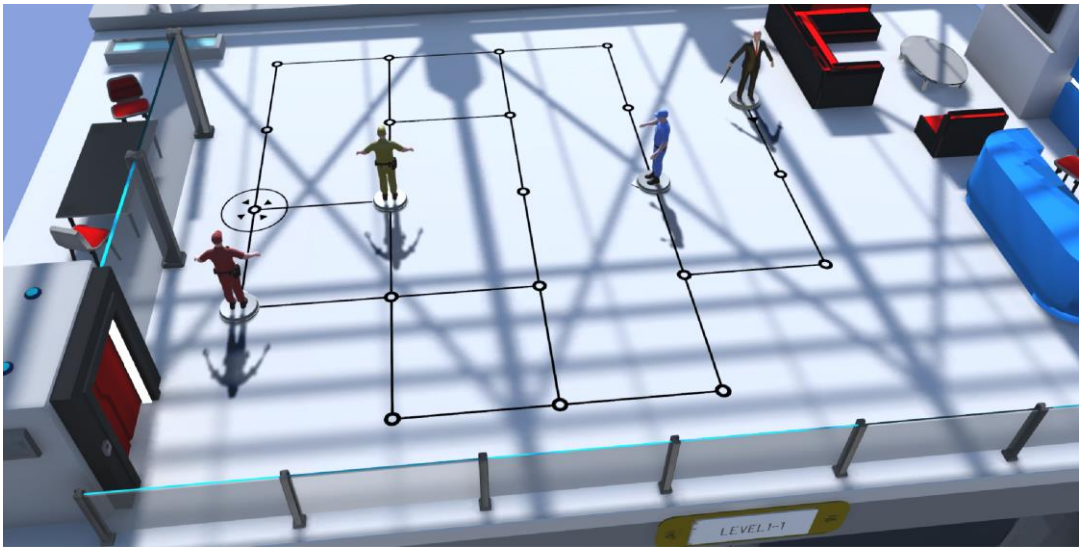
Cijelokupan proces izrade aplikacije sastoji se od sljedećih koraka:

1. Dizajn u C# komponentama
2. Animacije napravljene u iTween-u
3. Reakcije karaktera na unos sa tipkovnice
4. Ponašanje protivničkih karaktera
5. Dizajniran UI
6. Korištenje UnityEvent-a za riješvanje složenijih problema
7. ULS (Unity's Lighting System) za prikazivanje levela igre

Rad je strukturiran na način da nakon uvoda slijedi objašnjenje o tehnologijama koje su se koristile prilikom izrade ove aplikacije, pravila igranja, način razvijanja same igre te na kraju sam zaključak.

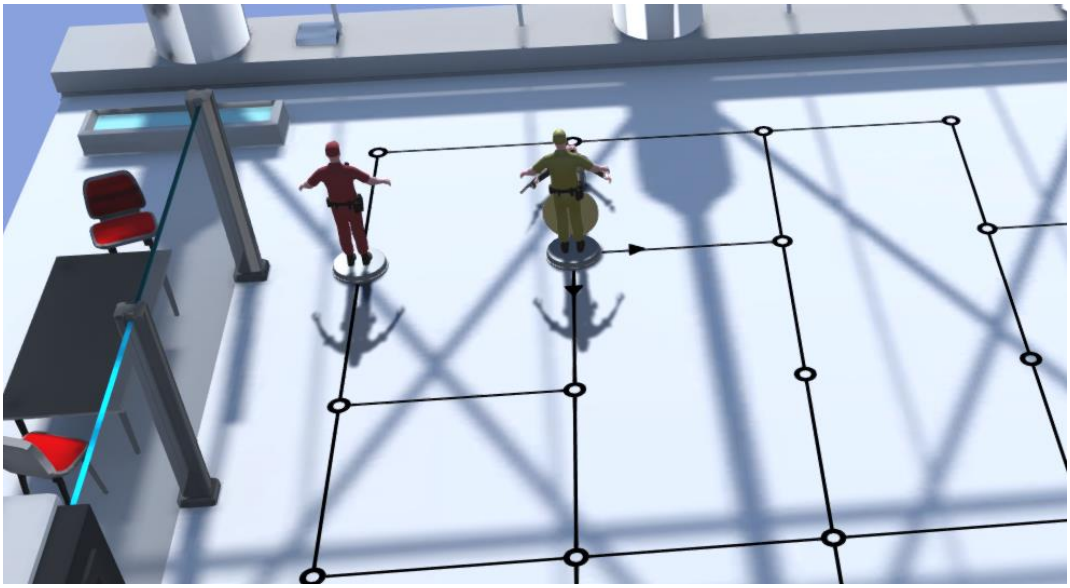
2. Pravila igre

Igrač ostvaruje pobjedu ako dođe do predviđenog polja koje označava cilj, a da u međuvremenu niti jedna protivnička figura nije okrenuta prema njemu.
Da bi se lakše došlo do pobjede, igrač je u mogućnosti eliminirati protivničke figure na način da dođe do polja na kojem se trenutno nalaze. Jednom kad igrač dođe do cilja, prelazi na sljedeću razinu, što se može uočiti na sljedećim slikama



Slika 2.1 - Početak prvog levela

Također, ako igrač ne bude dovoljno pažljiv, postoje mogućnosti da bude eliminiran od strane protivničkih figura na način da su figure prilikom njegove kretnje okrenute u smjeru njegovog dolaska.


Slika 2.2 – Trenutak poraza igrača

Igra se sastoji od dvije razine, prve koja je uvodna te za cilj ima podučiti igrača pravilima I načinu igre, te druge koja je nešto teža I predstavlja izazov.

3. Tehnologije

3.1 C#

Programski jezik C# (C sharp) je proizvod tvrtke Microsoft i nastao je kao odgovor na nedostatke postojećih jezika kao što su C, C++ i Visual Basic, u isto vrijeme kombinirajući njihove dobre strane. C# je potpuno objektno-orijentirani programski jezik, što znači da svi elementi unutar njega predstavljaju objekt. Objekt predstavlja strukturu koja sadrži podatkovne elemente, kao i metode i njihove međusobne interakcije.

Slika 3.1 – C-sharp

Prednosti C# nad C++:

C# ima takozvani „garbage collector“ koji olakšava rad s memorijom računala, dok se programeri C++ s istom mogu baviti izravno. C# također ima direktan pristup memoriji, no za razliku od C++ ne podržava makro-naredbe. C++ se odlično snalazi s ostalim libraryjima pisanima u C, dok je kod C# takav slučaj više izuzetak nego pravilo. Ovo su tek neke od razlika.

3.2 Unity

Unity je više platformni game engine, odnosno sustav za kreiranje i razvoj video igara, koji sadrži i razvojno okruženje, a razvijen je od tvrtke Unity Technologies. Unity omogućuje razvoj video igara za Unity web player, desktop i mobilne te razne igrače konzole.

Slika 3.2 – Unity Engine Overview

Unity kao game engine omogućuje kreiranje i razvoj 2D i 3D video igara i vrste igara poput FPS (First-Person Shooter), RTS (Real-Time Strategy), Trkače igre, TPS (Third-Person Shooter), RPG (Roleplaying game) itd. Sve te igre, zbog raznih mogućnosti i dodavanja modela te raznih specijalnih elemenata, svjetla, animacija i fizičkih karakteristika objektima, danas djeluju potpuno realistično. Unity Pro verzija je verzija Unity game engine-a koja daje potpunu funkcionalnost alata sa dodatno uključenim resursima, odnosno modelima koji se koriste za kreiranje i razvoje video igara. Ona dolazi sa podrškom za Web i desktop platformu a omogućuje i dodavanje raznih mobilnih platformi.

3.3 ITween

iTween je jednostavan i moćan alat za kreiranje animacijskih sustava kod Unity-a. Usredotočujući se na uspostavljena rješenja i okvire projekata kao što su TweenLite, Tweener i drugi sustavi, tweening i drugi interpolacijski sistemi se temeljene na Flashu, iTween je "battle-tested" rješenje za racionalizaciju proizvodnje u Unity okruženju.

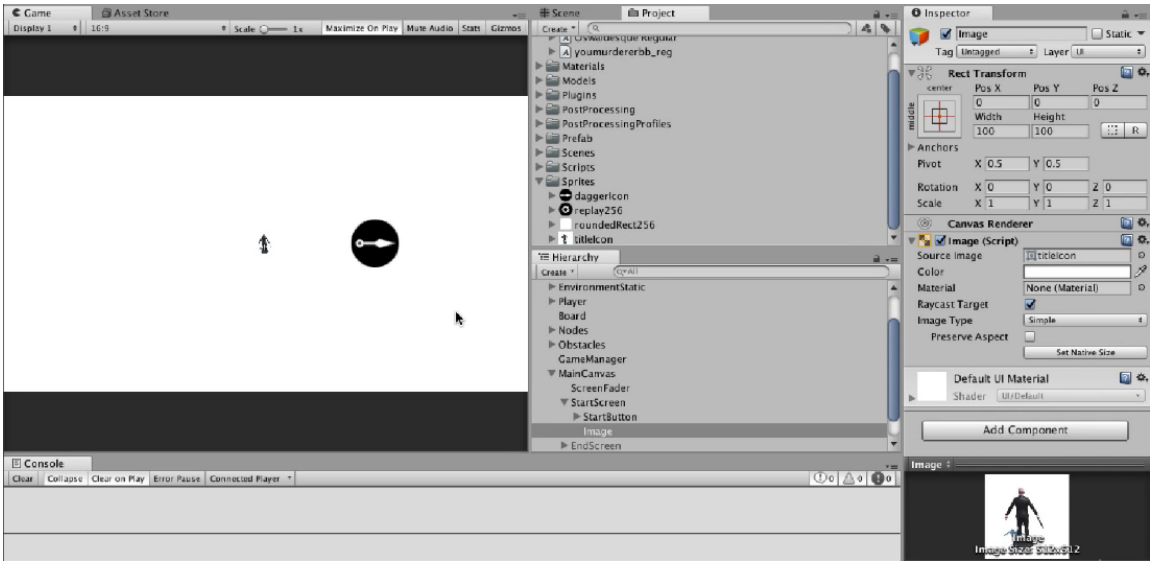
Slika 3.3 – iTween Framework

iTween je jedna C# datoteka koja se može koristiti s bilo kojim od programskih jezika koje Unity podržava, kao i sve verzije Unity-a.

4. Razvijanje Aplikacije

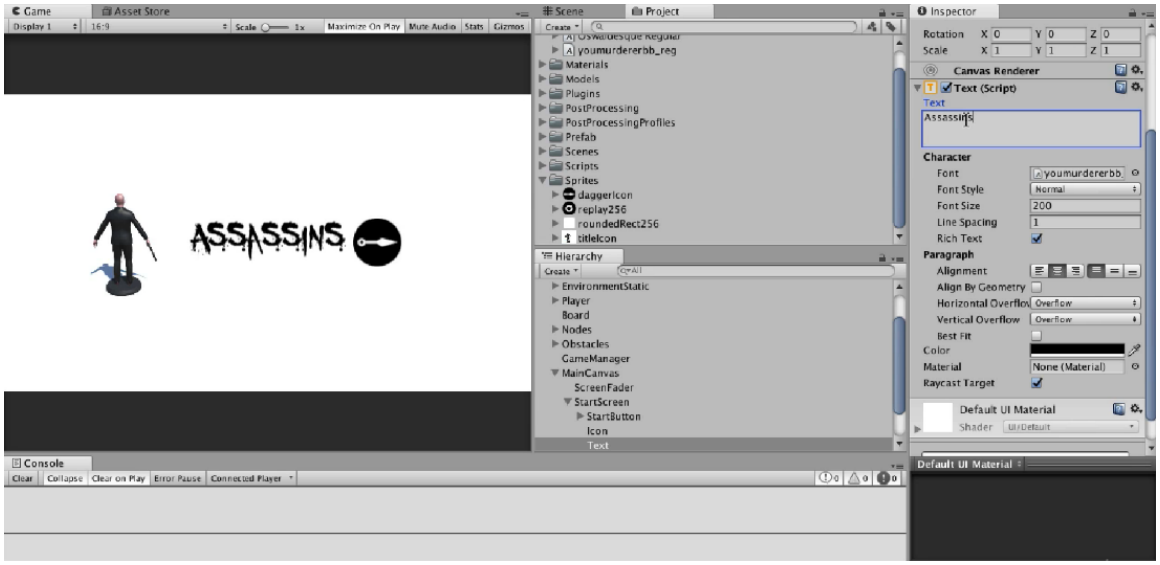
4.1 Početni prikaz

Kako bi se napravio čisti početni prikaz potrebno je sakriti (disable) ostale elemente te početi sa izgradnjom početnog prikaza. Slika 3.2 prikazuje čistu pozadinu sa dvije ikone koje u tom trenutku nisu dobro pozicionirane, no korištenjem X, Y, Z osi te “width” I “height” svojstava se to ispravlja što se vidi na sljedećim slikama.



Slika 4.1 – Početni Prikaz

Također, da bi se ubacio tekst, potrebno ga u prikazu dodati.
Važno je naglasiti da se fontovi teksta mogu po želji birati ne vezano za Unity.



Slika 4.2 – Biranje Fonta

4.2 Kreiranje levela

Kako bi smo počeli graditi igru prvo moramo kreirati novu scenu na kojoj ćemo raditi kroz ovaj projekt. U ovom slučaju koristimo dvije scene “Level1-1” i “Level1-2” unutar kojih postoje određeni grafički elementi kojima ćemo pridodjelti neku sliku te je prilagoditi da izgleda kao “sprite”.

Scene – Scene je spremnik koji sadrži sve komponente (Sprite, Label, Node, …) koji su potrebni igri te se u njoj mogu pojaviti. Scene je zadužen za odvijanje logike igrice te iscrtavanje sadržaja kao 1 frame u jedinici vremena.

Sprite – Sprite je objekt koji predstavlja 2D sliku koja može biti animirana ili transformirana izmjenom njezinih svojstava. U osnovi sve igre imaju najmanje jedan ili više Sprite objekata koji predstavljaju različite segmente igre.

Renderer – pojednostavljeno definirano renderer je odgovoran za prikaz sadržaja na ekranu. Za potrebe ovog rada nije potrebno ulaziti dublje u terminologiju i rad renderer-a.

4.3 Animacije grafičkog sučelja

Kako bi se omogućio elegatniji dizajn prilikom pokretanja aplikacije animirani su razni botuni pomoću scripte “GraphicMover.cs” u kojoj je definirana nova klasa koja omogućava ovu funkcionalnost. Ona omogućava da se sve animacije na početku I na kraju.

Klasa je realizirana na sljedeći način:

```
// create null objects to store the beginning and ending transform if none is specified
private void Awake()
{
    if (endXform == null)
    {
        endXform = new GameObject(gameObject.name + "XformEnd").transform;

        endXform.position = transform.position;
        endXform.rotation = transform.rotation;
        endXform.localScale = transform.localScale;
    }

    if (startXform == null)
    {
        startXform = new GameObject(gameObject.name + "XformStart").transform;

        startXform.position = transform.position;
        startXform.rotation = transform.rotation;
        startXform.localScale = transform.localScale;
    }

    Reset();
}
```

Slika 4.3 – GraphicMover.cs klasa inicijalni poziv

```
60      // reset the transform to starting values
61      public void Reset()
62      {
63          switch (mode)
64          {
65              case GraphicMoverMode.MoveTo:
66                  if (startXform != null)
67                  {
68                      transform.position = startXform.position;
69                  }
70                  break;
71              case GraphicMoverMode.MoveFrom:
72                  if (endXform != null)
73                  {
74                      transform.position = endXform.position;
75                  }
76                  break;
77              case GraphicMoverMode.ScaleTo:
78                  if (startXform != null)
79                  {
80                      transform.localScale = startXform.localScale;
81                  }
82                  break;
83          }
84      }
```

Slika 4.4 – GraphicMover.cs klasa resertiranje pozicije

```
86      // scale/rotate/translate the graphic depending on mode
87      public void Move()
88      {
89          switch (mode)
90          {
91              case GraphicMoverMode.MoveTo:
92                  iTween.MoveTo(gameObject, iTween.Hash(
93                      "position", endXform.position,
94                      "time", moveTime,
95                      "delay", delay,
96                      "easetype", easeType,
97                      "looptype", loopType
98                  ));
99                  break;
100             case GraphicMoverMode.ScaleTo:
101                 iTween.ScaleTo(gameObject, iTween.Hash(
102                     "scale", endXform.localScale,
103                     "time", moveTime,
104                     "delay", delay,
105                     "easetype", easeType,
106                     "looptype", loopType
107                 ));
108                 break;
109             case GraphicMoverMode.MoveFrom:
110                 iTween.MoveFrom(gameObject, iTween.Hash(
111                     "position", startXform.position,
112                     "time", moveTime,
113                     "delay", delay,
114                     "easetype", easeType,
115                     "looptype", loopType
116                 ));
117                 break;
118         }
119     }
120
```

Slika 4.5 – Gibanje grafičkih elemenata uz pomoć funkciskih poziva

Ovim načinom je ostvreno intuitivno integriranje scripte u razne elemente unutar projekte te smo ovim načinom jednostavno ostvarili razne načine gibanja elemenata kao na primjer pulsiranje ili translativno gibanje.
Koristeći “MoveTo” ili “MoveFrom” mode možemo s lakoćom napraviti klon nekog objekta na ekranu te mu maknuti sva svojstva i maknuti mu sva nepotreba svojstva pa ga staviti izvan vidljivog ekrana, iskoristit “MoveTo” ili “MoveFrom” početne lokacije ili kranje lokacije da dobijemo žljeno kretanje pri određenom trenutku u aplikaciji.

4.4 Kontrolna tijeka aplikacije

Za kontroliranje početka, trajanja i kraja igre koristi se scripta „GameManager.cs“
Unutar ove scripte također imamo razne privatne boolean varijable kojima kontroliramo put igre kao na primjer „hasLevelStarted“ te referenciramo naš board element na kojem smo napravili čvorove te linije kojima ih povezujemo.

```
You, 21 hours ago | 1 author (You)
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Events;
using UnityEngine.SceneManagement;
using System.Linq;

[System.Serializable]
8 references
public enum Turn
{
    4 references
    Player,
    2 references
    Enemy
}

3 references | You, 21 hours ago | 1 author (You)
public class GameManager : MonoBehaviour
{
    // reference to the GameBoard
    5 references
    Board m_board;

    // reference to PlayerManager
    8 references
    PlayerManager m_player;

    4 references
    List<EnemyManager> m_enemies;

    5 references
    Turn m_currentTurn = Turn.Player;
    1 reference
    public Turn CurrentTurn { get { return m_currentTurn; } }

    // has the user pressed start?
    4 references
    bool m_hasLevelStarted = false;
    0 references
    public bool HasLevelStarted { get { return m_hasLevelStarted; } set { m_hasLevelStarted = value; } }

    // have we begun gamePlay?
    3 references
    bool m_isGamePlaying = false;
    0 references
    public bool IsGamePlaying { get { return m_isGamePlaying; } set { m_isGamePlaying = value; } }

    // have we met the game over condition?
    5 references
    bool m_isGameOver = false;
    1 reference
    public bool IsGameOver { get { return m_isGameOver; } set { m_isGameOver = value; } }
```

Slika 4.6 – GameManager.cs klasa - Inicijalizacija

Uz pomoć te skripte napravljene su „Coroutine“ „StartLevelRoutine“, „PlayLevelRoutine“, „EndLevelRoutine“. Uz pomoć tih funcionalnosti omogućili smo da se u pojedinim trenutcima proizvoljno pokreće početak igre, završavaju leveli te time pokreću novi.

```
0 references
void Start()
{
    // start the main game loop if the PlayerManager and Board are present
    if (m_player != null && m_board != null)
    {
        StartCoroutine("RunGameLoop");
    }
    else
    {
        Debug.LogWarning("GAMEMANAGER Error: no player or board found!");
    }
}

// run the main game loop, separated into different stages/coroutines
0 references
IEnumerator RunGameLoop()
{
    yield return StartCoroutine("StartLevelRoutine");
    yield return StartCoroutine("PlayLevelRoutine");
    yield return StartCoroutine("EndLevelRoutine");
}

// the initial stage after the level is loaded
0 references
IEnumerator StartLevelRoutine()
{
    Debug.Log("SETUP LEVEL");
    if (setupEvent != null)
    {
        setupEvent.Invoke();
    }

    Debug.Log("START LEVEL");
    m_player.playerInput.InputEnabled = false;

    while (!m_hasLevelStarted)
    {
        //show start screen
        // user presses button to start
        // HasLevelStarted = true
        yield return null;
    }

    // trigger events when we press the StartButton
    if (startLevelEvent != null)
    {
        startLevelEvent.Invoke();
    }
}
```

Slika 4.7 – GameMenager.cs klasa - rutina za pokretanje levela

```
// gameplay stage
0 references
IEnumerator PlayLevelRoutine()
{
    Debug.Log("PLAY LEVEL");
    m_isGamePlaying = true;
    yield return new WaitForSeconds(delay);
    m_player.playerInput.InputEnabled = true;

    // trigger any events as we start playing the level
    if (playLevelEvent != null)
    {
        playLevelEvent.Invoke();
    }

    while (!m_isGameOver)
    {
        // pause one frame
        yield return null;

        // check for level win condition
        m_isGameOver = IsWinner();

        // check for the lose condition
    }
    // Debug.Log("WIN! ========================");
}

1 reference
public void LoseLevel()
{
    StartCoroutine(LoseLevelRoutine());
}

// trigger the "lose" condition
1 reference
IEnumerator LoseLevelRoutine()
{
    // game is over
    m_isGameOver = true;

    // wait for a short delay then...
    yield return new WaitForSeconds(1.5f);

    // ...invoke loseLoveEvent
    if (loseLevelEvent != null)
    {
        loseLevelEvent.Invoke();
    }

    // pause for two seconds and then restart the level
    yield return new WaitForSeconds(2f);

    Debug.Log("LOSE! ============================");

    RestartLevel();
}
```

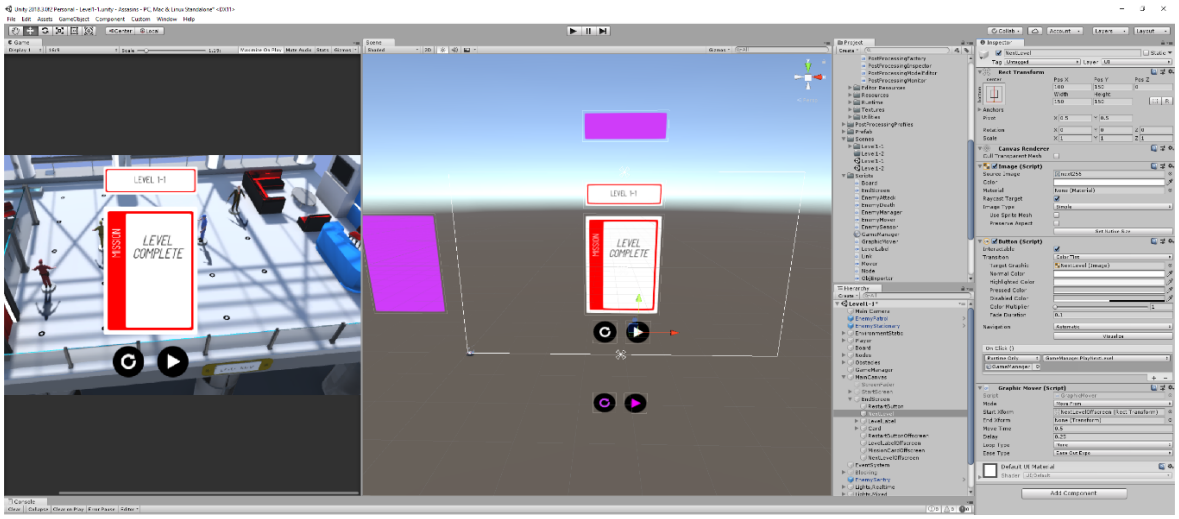
Slika 4.8 – GameMenager.cs klasa - rutina pri porazu u levelu

Ovdje možemo vidjeti primjer funkcije koju pozivamo kad glavni korisniči igrač izgubi onda pozovemo „LoseLevel“ funkciju koja pričeka neko vrijeme da se level ne završi iznenada te time poboljšamo igrajuće iskustvo korisniku.
Ova funkcija takodjer omogućava da mozemo ponovno pokrenuti level nakon sto izgubimo u igrici.

```
// restart the current level
2 references
void RestartLevel()
{
    Scene scene = SceneManager.GetActiveScene();
    SceneManager.LoadScene(scene.name);
}

// attach to StartButton, triggers PlayLevelRoutine
0 references
public void PlayNextLevel()
{
    Scene scene = SceneManager.GetActiveScene();
    SceneManager.LoadScene("Level1-2");
}

// attach to StartButton, triggers PlayLevelRoutine
0 references
public void PlayLevel()
{
    m_hasLevelStarted = true;
}

// has the player reached the goal node?
1 reference
bool IsWinner()
{
    if (m_board.PlayerNode != null)
    {
        return (m_board.PlayerNode == m_board.GoalNode);
    }
    return false;
}
```

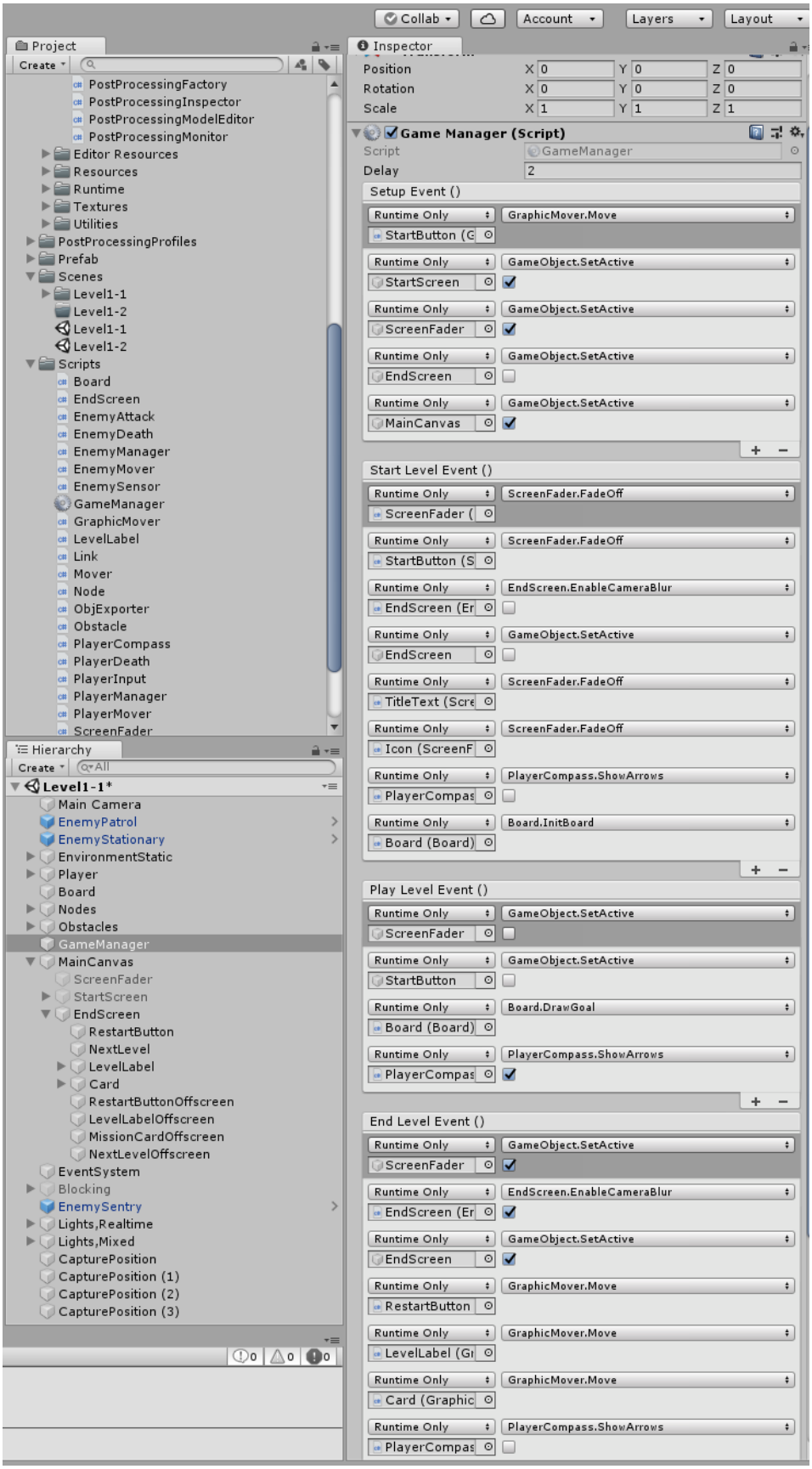
Slika 4.9 – GameMenager.cs klasa - funcije za pokretanje sljedećeg levela

Jako važan dio koda što omogučuje povezanost izmedju različitih scena („scenes“) u Unity-u jesu funkcije „RestartLevel“ te „PlayNextLevel“ koje smo poveali na elemente unutar naše postojuče scene u igrici.
Kako bi napredovali na sljedeći level u sceni „Level1-1“ napravili smo objekte kojima smo pridodjeli neku grafičku sliku kao na primjer „RestartButton“ i „NextLevel“ (Sprites) koji se pojave kada završimo level.
Nakon što se završi level pozovu se grafički elementi koji su sakriveni dok se boolean vrijednost kraja igre ne ostvari.



Slika 5.1 – GameMenager objekt unutar scene

Kada se taj događaj ostvari te se detektira da je došlo do kraja igre, „GameManager.cs“ scripta koja je dosada bila u beskojačnoj „while“ petlji se dovodi kraju te se poziva „EndLevelRoutine“ funkcija koja poziva unutar objekta „GameManager“ u toj specifičnoj sceni u ovom slučaju „Level1-1“

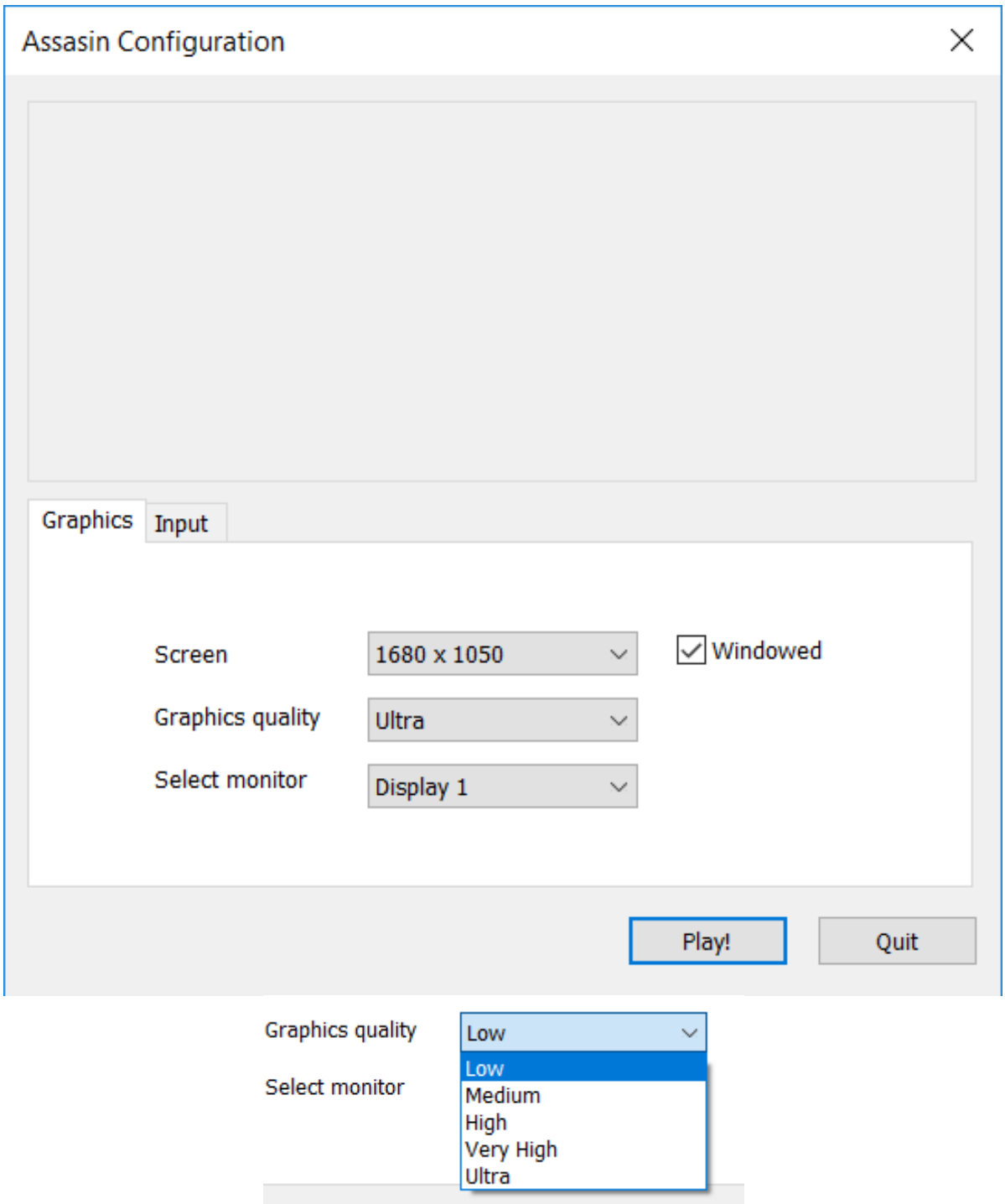


Slika 5.2 – GameMenager object opcije te odabrane akcije u pojedinom trenutku

Ovdje možemi vidjeti kako se točno pozivaju i postavljaju pojedini elementi u sceni pri početku pokretanja igrice, prilikom aktivnog igranja te kraja scene.

4.5 Konfiguracija grafike

Prilikom pokretanja aplikacije igraču je omogućeno da izabere rezoluciju i kvalitetu grafike za prikaz na ekranu. Igra može biti pokrenuta u punom prikazu ili u “Window” modu.



Slika 5.3 – Konfiguracija grafike

5. Zaključak

U ovom seminarskom radu prikazan je postupak izrade 3D desktop igre/aplikacije naziva Assasins. Assasins je trodimenzionalna igra na ploči u kojoj je cilj doći do prethodno definirane lokacije na ploči pri čemu je moguće eliminirati “protivnike” na način da se dođe do pozicije na kojoj se oni nalaze u trenutku dok nismo u njihovom fokusu.

Unity, C# te ITween su odlične tehnologije za izradu igrica koje su korištene vec jako dugo te uz njihovu popularnost dosta ravijene te se uz malo vremena I uloženog truda lako mogu izraditi zanimljiva I efektima riješenja. Navedene tehnologije su izuzetno intuitivne za korištenje te se bez prevelikog predznanja brzo savladaju do zadovoljavajuće razine znanja.

Unatoč velikom broju igara koje koriste sličnu logiku I pravila igre na ploči, Assasins je igra koja ipak donosi neko novo iskustvo I pozitivne dojmove.

LITERATURA

(1) https://assetstore.unity.com/packages/tools/animation/itween-84
(2) https://docs.unity3d.com/ScriptReference/UI.Image.html
(3) https://docs.microsoft.com/en-us/dotnet/csharp/
(4) https://assetstore.unity.com/categories/3d/characters
(5) https://www.photoshop.com/products
(6) https://www.irfanview.com/