UNIWERSYTET TECHNOLOGICZNO-PRZYRODNICZY
im. Jana i Jędrzeja Śniadeckich w Bydgoszczy

WYDZIAŁ TELEKOMUNIKACJI, INFORMATYKI
I ELEKTROTECHNIKI

PROJEKTOWANIE I PROGRAMOWANIE GIER - PROJEKT

na kierunku Informatyka Stosowana

Corpper

Pracę wykonał:

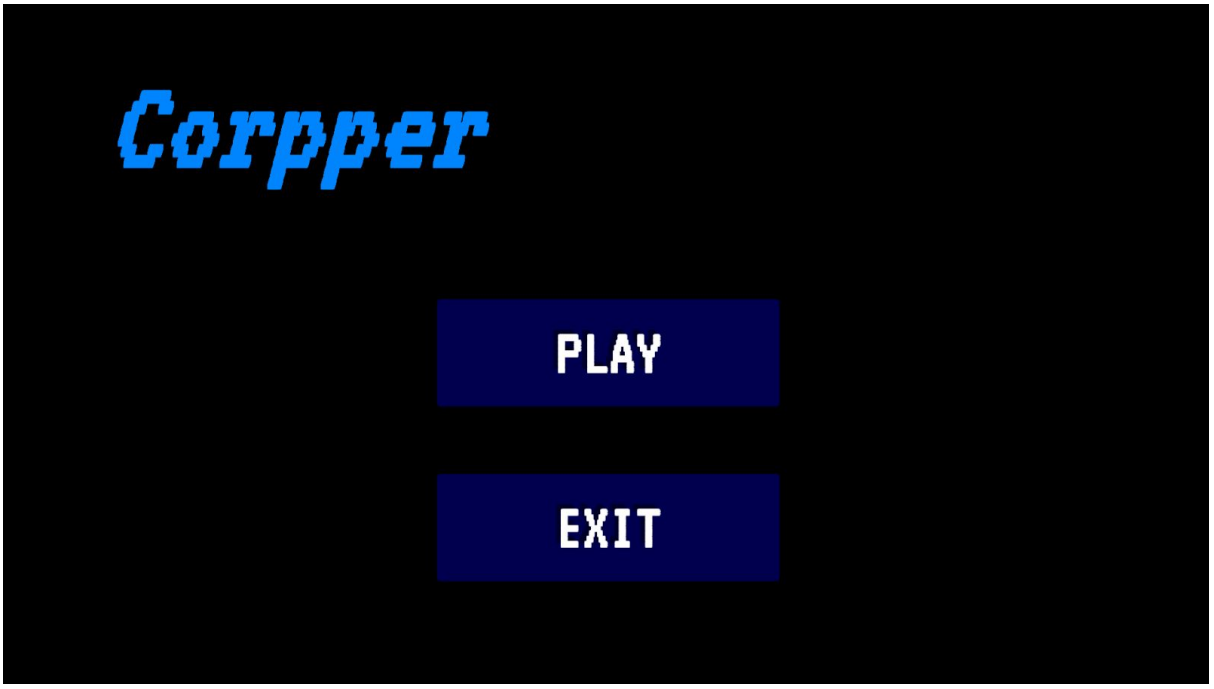
Piotr Lipkowski, Krystian Ryśnik

Bydgoszcz, czerwiec 2019

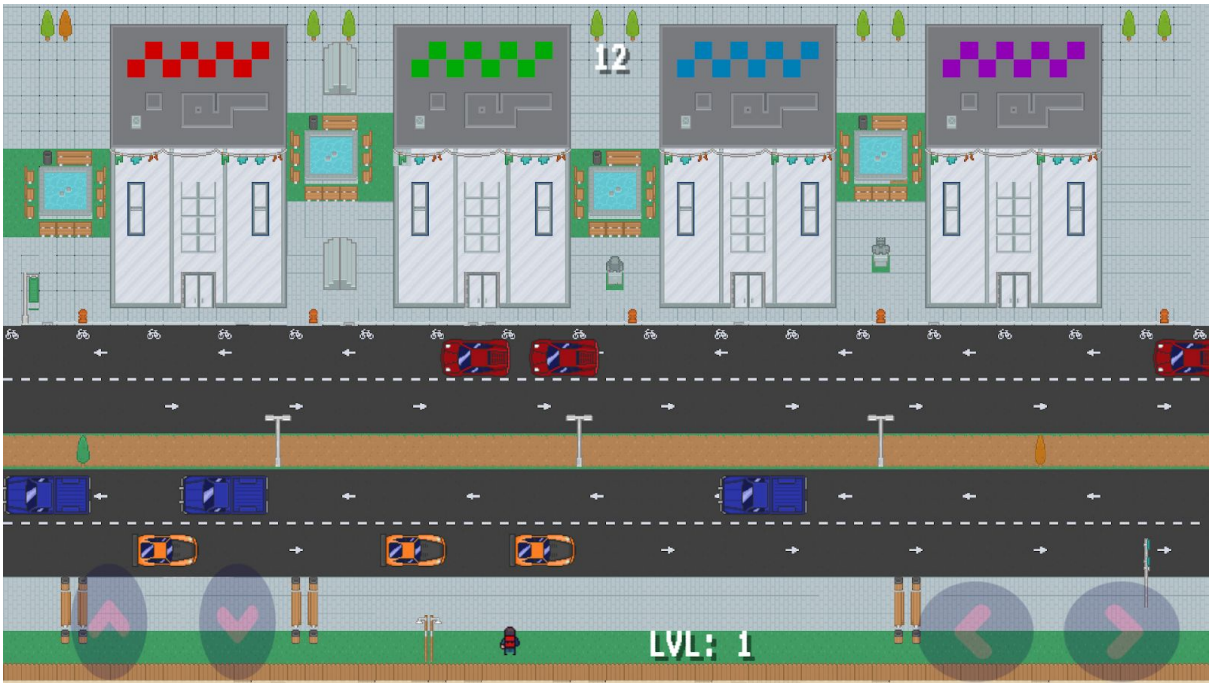
1. Ogólny opis gry

Zaprojektowana przez nas gra to zręcznościówka typu cross-road. Zadaniem gracza jest doprowadzenie odpowiednich postaci do wyznaczonego celu w tym przypadku budynków korporacji. Na drodze gracza do celu stają takie przeszkody jak poruszające się samochody oraz inne elementy środowiska takie jak ławki, kosze na śmieci, drzewa, sygnalizacja.

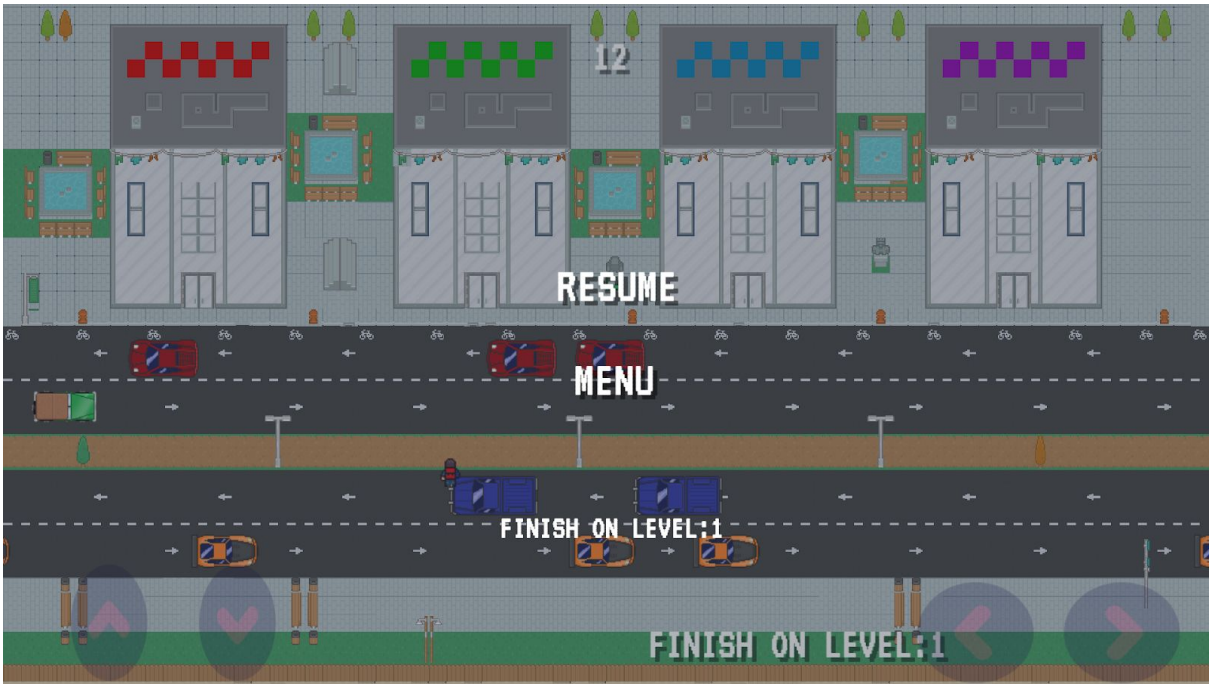
2. Zrzuty ekranów.



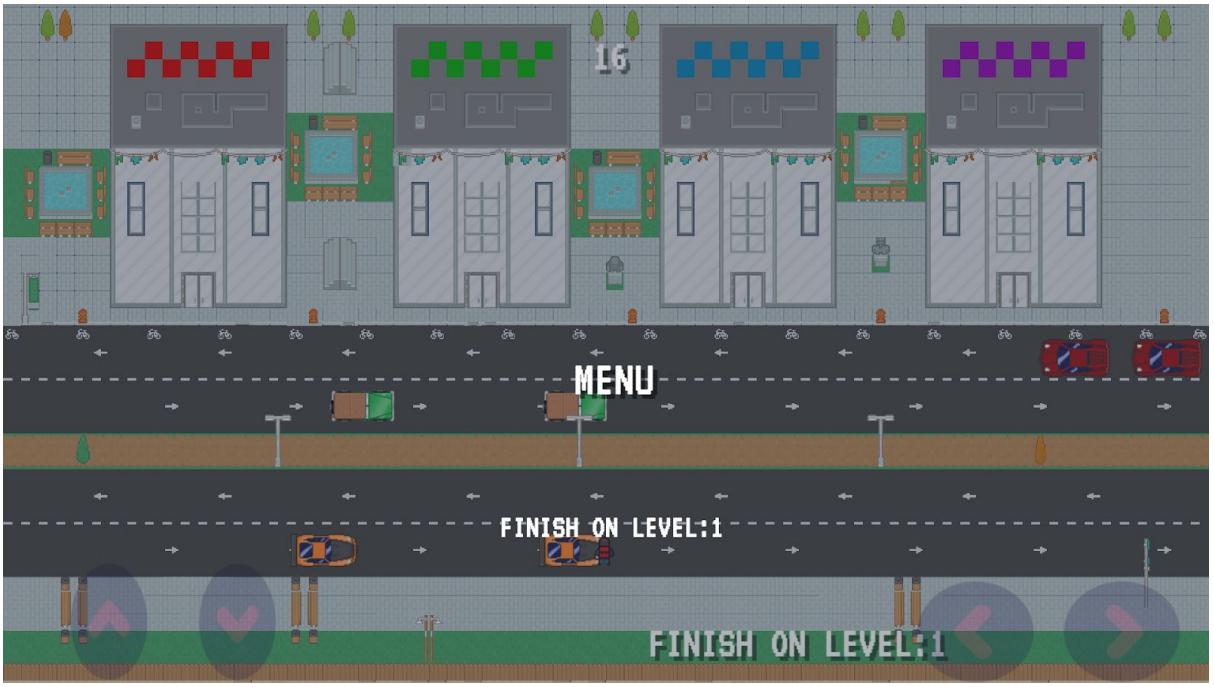
Ekran startowy gry



Ekran rozgrywki



Ekran porażki z możliwą kontynuacją.



Ekran porażki bez możliwej kontynuacji.

3. Cel gry.

Celem gry jest dotarcie do wszystkich czterech korporacji ich różnymi pracownikami. Z punktu widzenia użytkownika będzie to także osiągnięcie jak najwyższego poziomu bez popełnienia błędu.

4. Ograniczenia czasowe.

Na początku rozgrywki gracz otrzymuje 20 sekund na ukończenie rozgrywki, jednak z osiągnięciem każdej dodatkowej korporacji jego limit czasowy zostaje powiększony o dodatkowe 7 sekund.

5. Przeszkody.

Główną przeszkodą w grze są 4 linie samochodów poruszających się w różnych kierunkach i o różnych prędkościach. Kolizja z jednym z nich kończy się śmiercią gracza i przerwaniem gry.

6. Granice systemu gry.

Możliwość poruszania się po obszarze gry jest ograniczona z każdej części ekranu poprzez kolizje, w momencie zetknięcia postaci gracza z ograniczeniem obszaru, gracz nie może przekroczyć wyznaczonej linii. Kontynuując grę w każdym innym możliwym kierunku.

7. Ograniczenia przed niepożądanym działaniem gracza.

Gra została zabezpieczona przed niepożądanym działaniem gracza, który mógłby chciał sprawdzić co się stanie jeśli aktualny pracownik korporacji trafi nie do swojego budynku. W takim przypadku aktualny pracownik trafi ponownie na pole startowe skąd będzie musiał ponownie rozpocząć próbę przedostania się do swojej korporacji. Jego progres w grze nie zostanie poprawiony a spowoduje to jedynie utratę jego czasu na odpowiednie wykonanie celu.

8. Kolejny poziom.

Kolejny poziom zostanie uruchomiony w momencie gdy gracz dotrze wszystkimi czterema pracownikami do **odpowiednich** korporacji przed osiągnięciem limitu czasowego czasu.

9. Koniec gry.

Gra kończy się gdy użytkownik zderzy się z jednym z pojazdów lub nie zdąży dotrzeć do odpowiednich korporacji w przeznaczonym limicie czasowym. Na ekranie gracza pojawi się menu pauzy.

10. Objaśnienie odpowiednich klas w grze.

```
public class Player : MonoBehaviour
{
      public Rigidbody2D rb;
      float speed = 700f;
      public static bool lost = false;


      public void toRight()
      {
      rb.velocity = new Vector2(speed * Time.fixedDeltaTime, 0);
      }

      public void toLeft()
      {
      rb.velocity = new Vector2(-speed * Time.fixedDeltaTime, 0);
      }

      public void velocityZero()
      {
      rb.velocity = Vector2.zero;
      }


      public void toUp()
      {
      rb.velocity = new Vector2(0, speed * Time.fixedDeltaTime);
      }

      public void toDown()
      {
      rb.velocity = new Vector2(0, -speed * Time.fixedDeltaTime);
      }




      private void OnTriggerEnter2D(Collider2D collision)
      {
      if(collision.tag == "Car")
      {
```

```
            Debug.Log("We lost");
            lost = true;
      }

      if(collision.tag == "Corpo1")
      {
            Destroy(gameObject);
      }


      }


}
```

Klasa Player jest odpowiedzialna za główne postacie kierowane przez użytkownika oprócz takich informacji w jakim kierunku i z jaką prędkością w danym momencie może poruszać się gracz zawiera zmienną lost, która przyjmuje wartość prawdziwą w momencie gdy użytkownik zetknie się z samochodem co jest równoznaczne z przegraniem gry. Klasa ta również wykrywa kolizje z właściwym budynkiem korporacji usuwając obiekt klasy Player z planszy.

```
public class Car : MonoBehaviour
{
      public Rigidbody2D rb;

      float speed = 1f;

      private void Start()
      {
      speed = 10f;
      }

      void FixedUpdate()
      {
      Vector2 forward = new Vector2(transform.up.x,
transform.up.y);
      rb.MovePosition(rb.position + forward * Time.fixedDeltaTime *
speed);
      Destroy(gameObject, 8.0f);
      }


}
```

Klasa Car zajmuje się przechowywaniem działań samochodu takim jak nadanie mu prędkości i kierunku (jazda do przodu).
Usuwa także samochód z mapy gdy przekroczył on wyznaczony limit czasowy istnienia w systemie gry. Ogranicza to ilość potrzebnych samochodów na planszy.

```
public class CarSpawner : MonoBehaviour
{
      public float spawnDelay = .3f;

      public GameObject car, car2,car3,car4;

      public Transform[] spawnPoints;

      float nextTimeToSpawn = 0f;

      private void Update()
      {
      if (nextTimeToSpawn <= Time.time)
      {
            SpawnCar();
            nextTimeToSpawn = Time.time + spawnDelay;
      }

      }

      void SpawnCar()
      {
      int randomIndex = Random.Range(0, spawnPoints.Length);
      Transform spawnPoint = spawnPoints[randomIndex];

      if(randomIndex == 0)
      {
            Instantiate(car, spawnPoint.position,
spawnPoint.rotation);
      }
      if(randomIndex == 1)
      {
            Instantiate(car2, spawnPoint.position,
spawnPoint.rotation);
      }
      if (randomIndex == 2)
      {
            Instantiate(car3, spawnPoint.position,
spawnPoint.rotation);
      }
      if (randomIndex == 3)
```

```
      {
            Instantiate(car4, spawnPoint.position,
spawnPoint.rotation);
      }

      }
```

Klasa CarSpawner zajmuje się pojawianiem się na mapie kolejnych samochodów w określonym przedziale czasowym. Jeśli zmienna **nextTimeToSpawn** jest mniejsza od aktualnego czasu gry to samochód ma się pojawić, a do aktualnej wartości tej zmiennej ma zostać przypisana nowa wartość, którą jest aktualny czas gry + zdefiniowany przedział czasowy. Aż do kolejnego spełnienia wyżej wymienionego warunku. Funkcja **SpawnCar** dodatkowo losuje w, której z tablic czyli punktów startowych dla auta ma się pojawić samochód. A funkcja **Instantiate** tworzy wskazany obiekt dla danej tablicy.

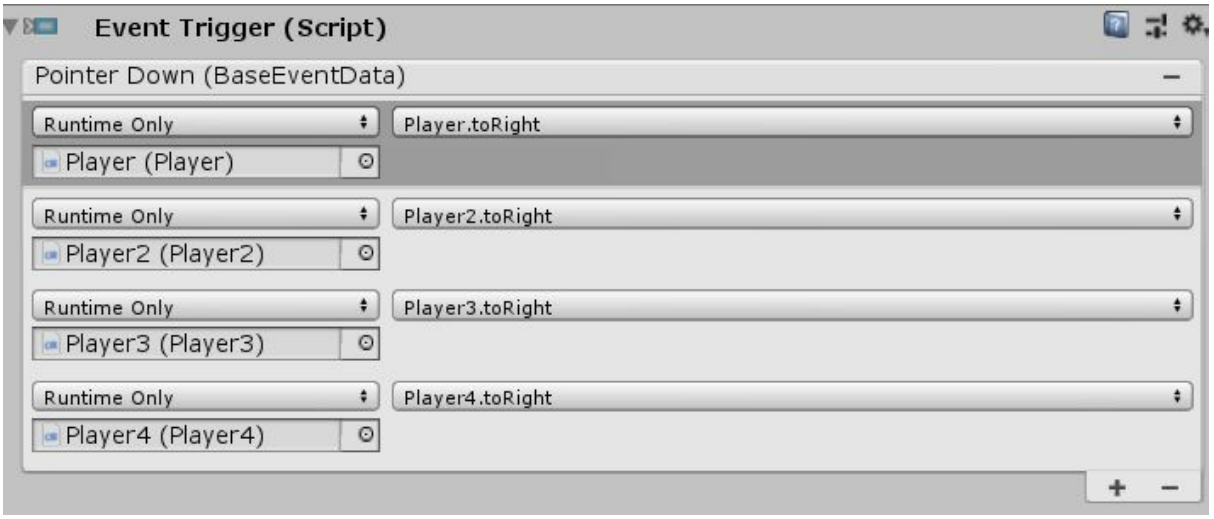
```
public class Corpo1 : MonoBehaviour
{
      public bool reachCorpo1 = false;
      public Corpo2 corpo2;
      public Corpo3 corpo3;
      public Corpo4 corpo4;

      public GameObject player2;
      public GameObject player3;
      public GameObject player4;

      public Timer timer;


      private void Start()
      {
      corpo2 = GameObject.Find("Corpo2").GetComponent<Corpo2>();
      corpo3 = GameObject.Find("Corpo3").GetComponent<Corpo3>();
      corpo4 = GameObject.Find("Corpo4").GetComponent<Corpo4>();
      }

      private void OnTriggerEnter2D(Collider2D collision)
      {
```

```
      if (collision.tag == "Player")
      {
            reachCorpo1 = true;
            player2.SetActive(true);
            timer.currentTime += 7f;
      }
      else if(collision.tag == "Player2")
      {
            player2.transform.position = new Vector3(-6.6f,
-16.63f, 0);
      }
      else if (collision.tag == "Player3")
      {
            player3.transform.position = new Vector3(11.9f,
-16.64f, 0);
      }
      else if (collision.tag == "Player4")
      {
            player4.transform.position = new Vector3(11.9f,
-16.64f, 0);
      }



      if (reachCorpo1 && corpo2.reachCorpo2 && corpo3.reachCorpo3
&& corpo4.reachCorpo4)
      {
            Debug.Log("You reach all");
            LevelScript.levelValue += 1;
            Debug.Log(LevelScript.levelValue);
      }

      }



}
```

Klasa Corpo1 zajmuje się przechowywaniem informacji na temat zależności osiągnięcia przez gracza danej korporacji w wskazanym przypadku pierwszej. Wskazuje, że jeśli tylko **Player** czyli pierwszy kierowany pracownik trafi do swojej korporacji to zmienna **reachCorpo1** ma zostawić ustawiona na prawdziwą a na planszy ma się pojawić kolejny pracownik a w klasie Timer zmienna odpowiedzialna za czas wymagany do ukończenia poziomu zostanie zwiększona o dodatkowe 7 sekund.. Klasa ta również tworzy zabezpieczenie przed wejściem do korporacji niewłaściwego aktualnie prowadzonego przez gracza pracownika. **Klasy zawierające dodatkowe numery czyli Corpo2, Corpo3, Corpo4** zawierają informacje o zwiększeniu licznika poziomów, gdy gracz osiągnie kolejny poziom czyli zaliczy wszystkie wskazane korporacje.

```
public class MainMenu : MonoBehaviour
{

      public static bool showResumeButton = true;
      public static bool usedAd = false;


      public void PlayGame()
      {
      LevelScript.levelValue = 1;
      Player.lost = false;
      Player2.lost = false;
      Player3.lost = false;
      Player4.lost = false;
      Timer.timeIsOver = false;
      Time.timeScale = 1f;
      showResumeButton = false;
      usedAd = false;
      SceneManager.LoadScene("Game");
      }

      public void QuitGame()
      {
      Application.Quit();
      }
}
```

Klasa MainMenu zajmuje się restartem zmiennych mogących być ustawionych na inne wartości w trakcie rozgrywki niż te, które są wymagane na początku rozgrywki.



Na ekranie rozgrywki pojawiają się przyciski używane do sterowania naszą postacią, na wyżej wyświetlanym obrazie widzimy sposób działania jednego z nich, gdy gracz go wciśnie to, u aktualnie aktywnej postaci zostanie wywołana metodą odpowiedzialna za ruch w prawo.

11. Opis działania mikropłatności w grze.

```
public class LoseMenu : MonoBehaviour
{

      public GameObject loseMenuUI;
      public Timer timer;
      public GameObject resumeButton;


      void Update()
      {

      if (Player.lost || Player2.lost || Player3.lost || Player4.lost ||
Timer.timeIsOver)
      {
            Pause();
      }

      }


      void Pause()
      {

      if (Application.internetReachability ==
NetworkReachability.NotReachable)
      {
            MainMenu.showResumeButton = false;
      }
      else if(Advertisement.IsReady()  &&
!(Application.internetReachability == NetworkReachability.NotReachable)
&& !MainMenu.usedAd)
      {
            MainMenu.showResumeButton = true;
      }

      if (MainMenu.showResumeButton)
      {
            resumeButton.SetActive(true);
      }

      if (!MainMenu.showResumeButton)
```

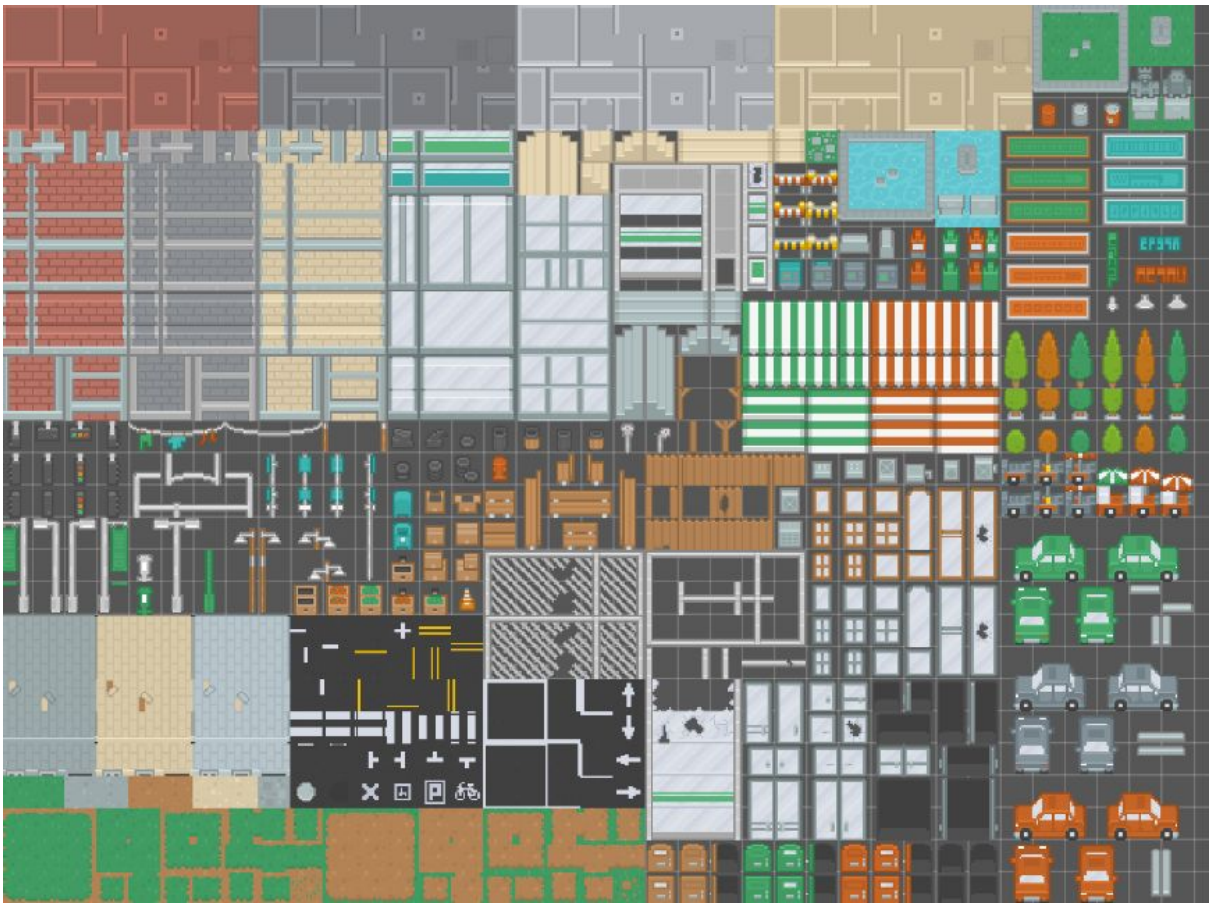
```
	{
		resumeButton.SetActive(false);
	}

	loseMenuUI.SetActive(true);
	Time.timeScale = 0f;
	}

	public void WatchAndPlay()
	{
	//watch ad and play
	Debug.Log("Show res" + MainMenu.showResumeButton);
	if (MainMenu.showResumeButton)
	{
		Advertisement.Show("rewardedVideo", new ShowOptions() {
resultCallback = HandleAdResult});
		MainMenu.usedAd = true;
	}
	else
	{
		Player.lost = false;
		Player2.lost = false;
		Player3.lost = false;
		Player4.lost = false;
		loseMenuUI.SetActive(false);


		Timer.timeIsOver = false;
		timer.Start();

		Time.timeScale = 1f;
		MainMenu.showResumeButton = true;
		MainMenu.usedAd = false;
		SceneManager.LoadScene("Menu");
	}



	}


	private void HandleAdResult(ShowResult result)
	{
	switch (result)
	{
```

```
            case ShowResult.Finished:
                  Debug.Log("Finished");
                  Player.lost = false;
                  Player2.lost = false;
                  Player3.lost = false;
                  Player4.lost = false;
                  loseMenuUI.SetActive(false);


                  Timer.timeIsOver = false;
                  timer.Start();

                  Time.timeScale = 1f;
                  MainMenu.showResumeButton = false;
                  MainMenu.usedAd = true;
                  SceneManager.LoadScene("Game");

                  break;
            case ShowResult.Skipped:
                  Debug.Log("Skipped");
                  SceneManager.LoadScene("Menu");
                  MainMenu.showResumeButton = true;
                  MainMenu.usedAd = false;

                  break;
            case ShowResult.Failed:
                  Debug.Log("Failed");
                  MainMenu.showResumeButton = true;
                  SceneManager.LoadScene("Menu");
                  MainMenu.usedAd = false;
                  break;
      }
      }


}
```

Realizacja mikropłatności w grze zachodzi poprzez wyświetlanie reklam w klasie **LoseMenu**. W trakcie działania aplikacji sprawdzane jest czy zaszedł, któryś z warunków takich jak porażka gracza lub koniec czasu. Jeśli wskazany warunek zachodzi pojawia się menu pauzy.

W funkcji **Pause** sprawdzane jest czy możliwe jest połączenie z internetem, które ma na celu umożliwić odtworzenie reklamy. Jeśli nie ma połączenia to przycisk odpowiedzialny za włączenie reklamy będzie wyłączony, jeśli jest połączenie a reklama jest gotowa do obejrzenie i nie była wcześniej używana w aktualnie rozgrywce to przycisk będzie widoczny.
Do wskazanego przycisku podłączona jest metoda **WatchAndPlay**, która jeszcze raz sprawdza czy przycisk jest aktualnie dostępny i umożliwia uruchomienie reklamy następnie ustawia wartość zmiennej **usedAd**, na prawdziwą, co oznacza, że reklama została już wykorzystana.
Samo włącznie reklamy zachodzi za pomocą funkcji **Show,** która jako swój argument przyjmuje nazwę reklamy do uruchomienia oraz wynik innej wywoływanej funkcji **HandleAtResult,** która zawiera instrukcję switch, która może zakończyć w 3 różnych wariantach gdy reklamy zostanie poprawnie obejrzana to zmienne odpowiedzialne za porażkę gracza zostają zresetowane bez względu na to jakim pracownikiem gracz popełnił błąd i przegrał. Zmienna **usedAd** zostaje ustawiona na prawdziwą co oznacza, że użytkownik na pewno obejrzał całą reklamę. A przycisk kontynuowania nie jest już dostępny a gracz może powtórzyć poziom, na którym przegrał. Drugim wariantem jest zachowanie funkcji, gdy gracz przewinął reklamę w naszym przypadku jest to powrót do menu głównego co nie dalej możliwości dalszej gry, trzeba ją zacząć od początku. To samo zachodzi w przypadku nie obejrzenia reklamy. Zmienna odpowiedzialna za obejrzenie reklamy zostaje ustawiona na fałszywą, ponieważ nie doszło do jej obejrzenia a gracz powróci do Menu głównego.

12. Realizacja grafiki.



Grafika w grze została zrealizowana w oparciu o TileMap każdy element o rozmiarach 16 x 16 px. Każdy element został dopasowany pod kątem tego czy gracz może się po nim poruszać czy jest też dla niego przeszkodą. Jeśli był on przeszkodą znalazł się w osobnej warstwie.