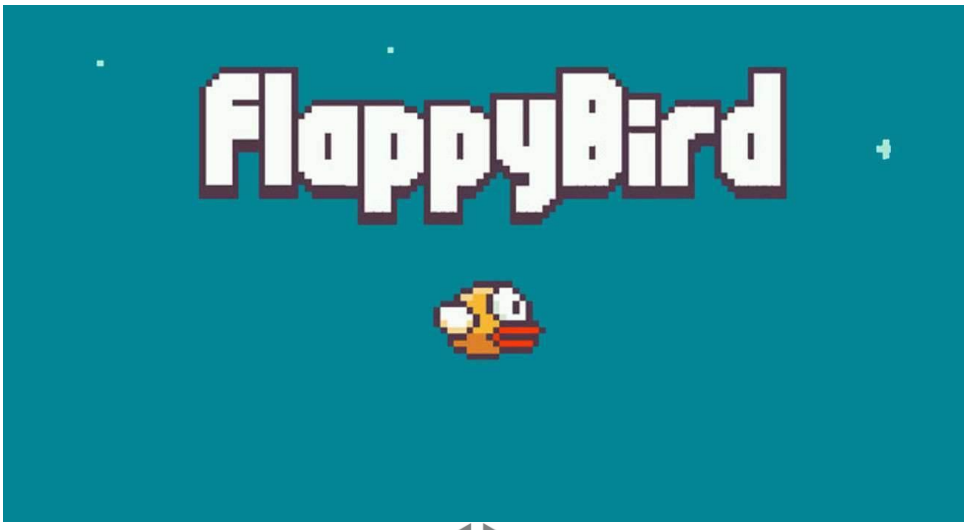
POLITECHNIKA BYDGOSKA

Autorzy: Karol Fac, Mariusz Jackowski

Numery indeksu: 111592, 113031

PROGRAMOWANIE APLIKACJI MOBILNYCH

DOKUMENTACJA PROJEKTU



Spis treści

1. Ogólny opis gry

Cel projektu

Celem projektu jest odwzorowanie gry Flappy Bird. Gra została udostępniona 24 maja 2013 roku. Twórcą gry jest Wietnamczyk Nguyễn Hà Đông. Flappy Bird jest tzw. „side-scrollerem”. Głównym celem jest omijanie rur wystających z dołu i z góry ekranu przez stukanie palcem w ekran telefonu. W styczniu 2014 roku gra była najczęściej ściąganą aplikacją na App Store.

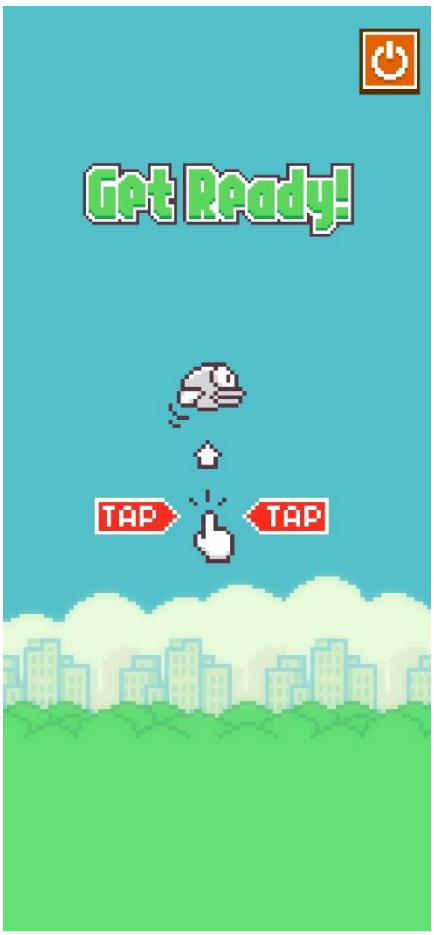
Użytkownicy, postacie, przedmioty

Gra została stworzona dla pojedynczego gracza, jednak możliwa jest rywalizacja między graczami poprzez uzyskanie większego wyniku podczas gry. Gracz wciela się w postać lecącego ptaka, którego zadaniem jest przelecenie między rurami zdobywając punkty przy tym nie wlatując w rury czy też ziemię.

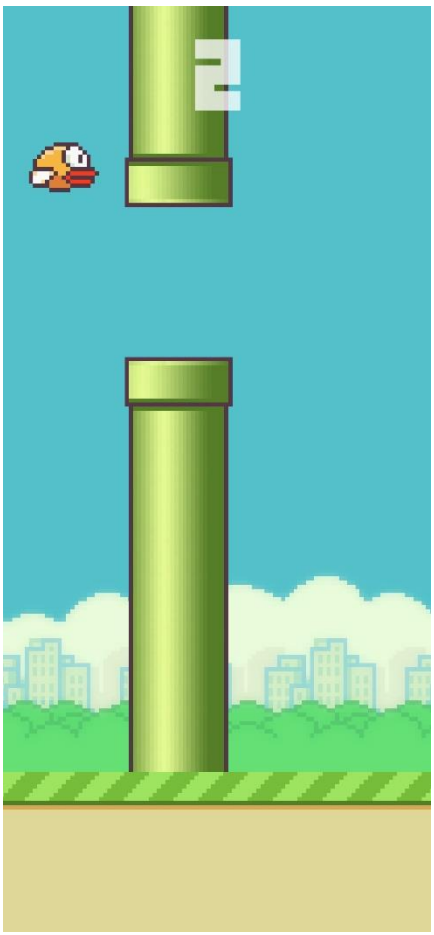
Granice systemu gry

Gra została zaprojektowana, by dopasowywała się do wielkości ekranu i działa w trybie pełnoekranowym. Wymuszana jest orientacja pionowa(portrait) i zablokowana jest pozioma(landscape).

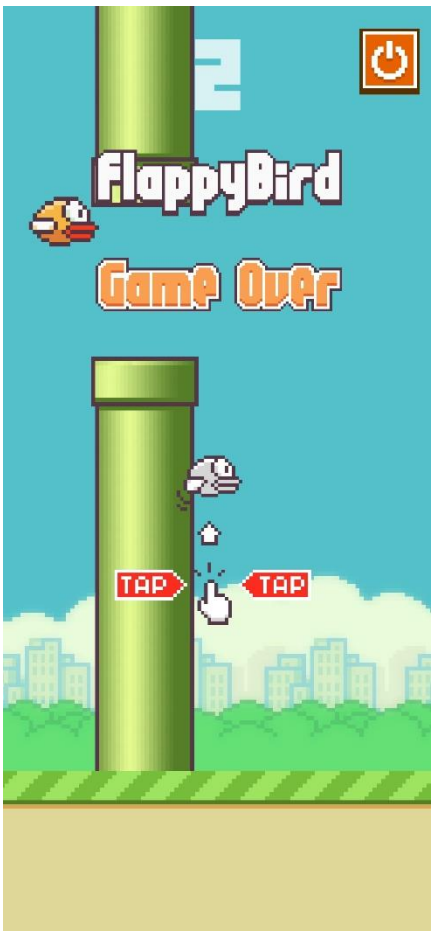
Dzięki takiemu ustawieniu gra wyświetla się odpowiednio i gra się najwygodniej. Gra dopasowuje się automatycznie do proporcji ekranu co pozwala nam grać na różnych urządzeniach. Grafiki w grze skalują się względem aktualnych wymiarów ekranu



Rysunek 1 - ekran startowy



Rysunek 2 - ekran rozrywki



Rysunek 3 - ekran końca gry

Sterowanie odbywa się za pomocą ekranu dotykowego. Przy każdym dotknięciu ekranu nasz bohater podlatuje w górę.

Lista możliwości

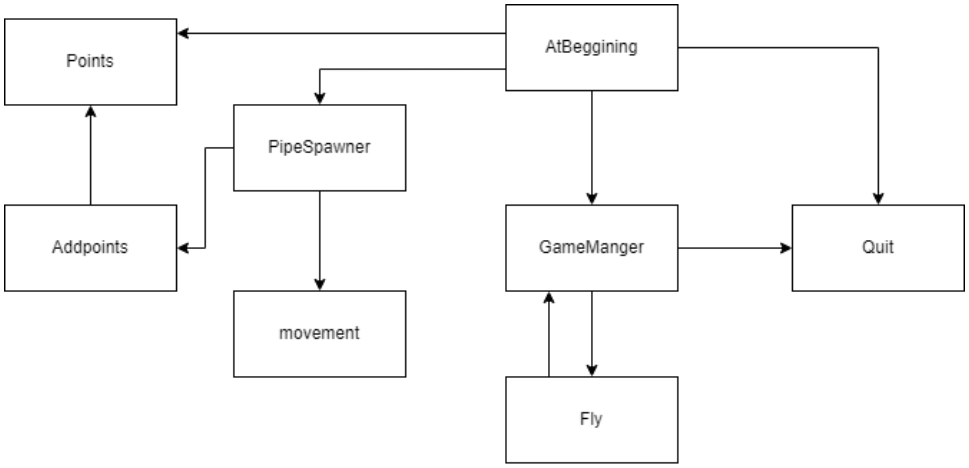
- Gracz:
  - Uruchamia grę,
  - Porusza się postacią,
  - Zdobywa punkty przelatując między rurami,
  - Uderza w przeszkodę uruchamiając ekran końca gry,
  - Po śmierci możliwość rozpoczęcia rozgrywki na nowo,
  - Wyjście z gry

- Rury:
  - Poruszają się w lewo poza widok gracza
  - Gdy gracz w nie uderzy dochodzi do śmierci bohatera

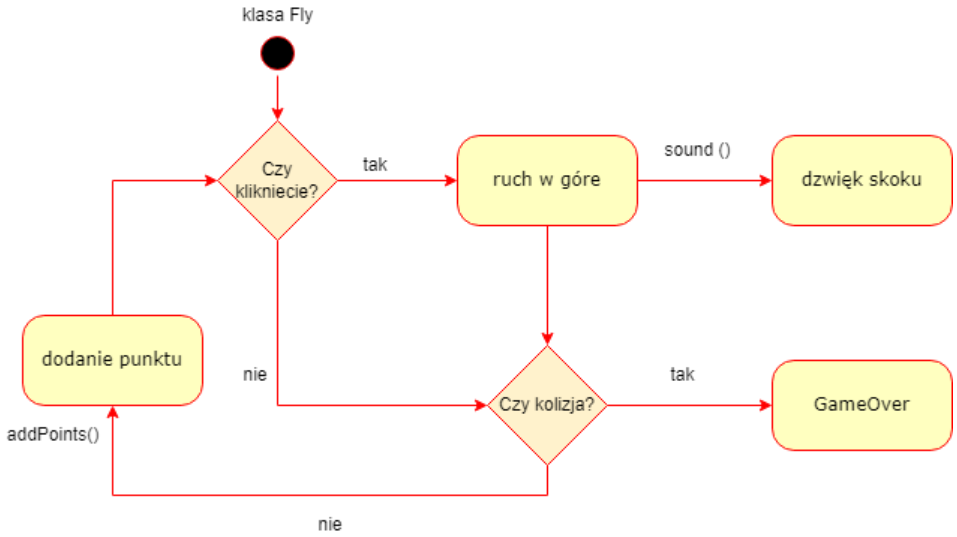
2. ANALIZA DZIEDZINY

- Klasy zidentyfikowane w dziedzinie, opis atrybutów
    - Klasa AtBeggining – Po kliknięciu na ekran ładuje główną scenę gry.
    - Klasa GameManager – Po załadowaniu sceny gry rozpoczyna ją, podczas kolizji z obiektem ładuje scenę końca gry oraz możliwość rozpoczęcia gry na nowo.
    - Klasa AddPoints – Przy przeleceniu między rurami odgrywa dźwięk zdobycia punktu i dodaje punkt.
    - Klasa Points – Na początku gry ustawia punktacje na 0 i aktualizuje punktacje na górze ekranu.
    - Klasa Fly - ustawiamy szybkość naszych podskoków/latania oraz przy każdym z nich odgrywamy odpowiedni dźwięk
    - Klasa PipeSpawner – przy uruchomieniu gry tworzy nowy obiekt rur z losowo umiejscowiona luką między nimi oraz ustawia czas między stworzeniem nowych oraz po jakim czasie zostaną zniszczone.
    - Klasa Movement - Ustawia prędkość poruszania się rur
    - Klasa Quit – Po kliknięciu przycisku wyłącza grę

- Diagram klas



- Diagram stanów



3. Specyfikacja wymagań

- Definicja przypadków użycia

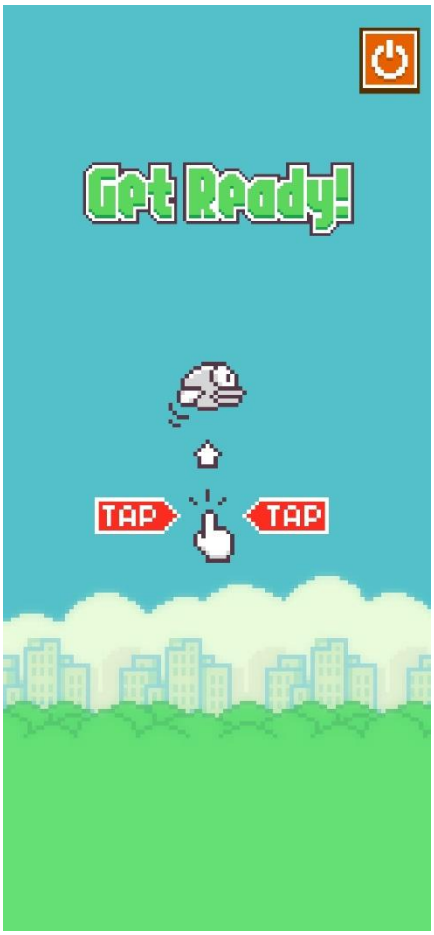
  Gracz po uruchomieniu ma możliwość wyboru wyjścia z gry po przez przycisk lub ją rozpocząć klikając na ekran.
  - Rozpoczęcie gry – po rozpoczęciu gracz rozpoczyna przygodę naszego spadającego ptaka. Jeżeli nie chcemy zakończyć od razu rozrywki musimy pomóc naszemu bohaterowi omijać rury poprzez przelatywanie między nimi, a tym samym zdobywamy punkty.
  - Śmierć – przy uderzeniu w jakąkolwiek przeszkodę lub wylecenie z granicę gry dochodzi do śmierci naszej postaci i przechodzimy do ekranu śmierci na którym możemy wrócić do gry lub wyjść z aplikacji.

4. Analiza i projekt

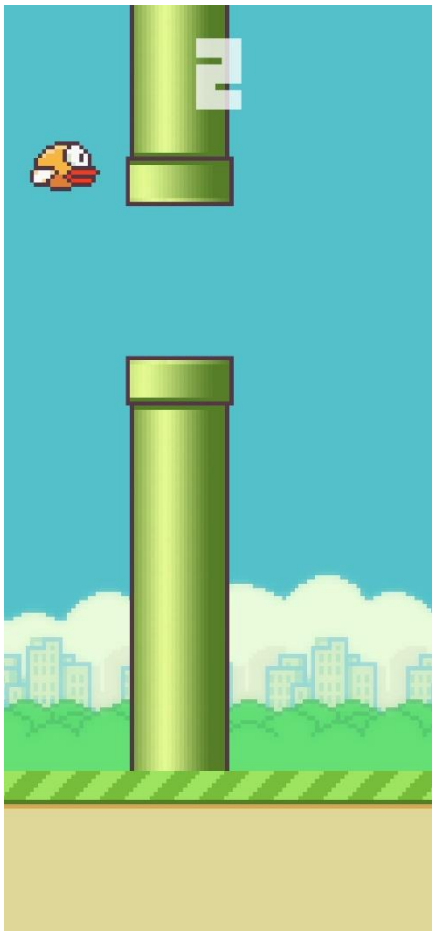
- Architektura systemu gry
  - Wyliczenie warstw: model, logika gry, interfejs użytkownika
    - **Model**
      Modelem systemu gry są wszystkie wykorzystywane klasy, obiekty i assety do przechowywania umieszczenia obiektów i ich ustawień. Takich jak miejsce tworzenia nowych rur, czas po jakim są usuwane, miejsce w jakim umieszczony jest bohater, prędkość animacji i efekty dźwiękowe. Rzeczy te budują całą naszą grę i interfejs użytkownika.
    - **Logika gry**
      Logika jest równie ważnym elementem projektu jakim jest model. To dzięki niej jesteśmy w stanie zobaczyć działanie całej naszej gry. Gracz widzi efekty interakcji postaci z światem oraz efekty sterowania postacią przez gracza. Mechaniki zaimplementowane:
      - postać opadająca automatycznie
      - skakanie/lot postaci
      - zdobywanie punktów
      - animacje odpowiadające za poruszanie się postaci(poruszająca się podłoga i rury)
      - miejsce tworzenia nowych rur
      - usuwanie po upływie czasu rur
      - efekty dźwiękowe dla skoku, zdobycia punktu i uderzenia w przeszkodę
    - **Interfejs użytkownika**
      Jest to część aplikacji, która widzi gracz. Nie obchodzi go przecież jak coś działa w kodzie tylko tego efekty. W grze występują 3 widoki:
      - główne menu w tym ekranie możemy wyjść przyciskiem z gry lub dotknąć ekranu i rozpocząć tym samym przechodząc do następnego widoku
      - ekran rozrywki jest to główny ekran na którym skaczemy naszą postacią, omijamy rury, zdobywamy punkty oraz giniemy tym samym przechodząc do ostatniego widoku
      - ekran śmierci ekran śmierci daje nam możliwość wyjścia z gry jak również rozpoczęcia jej na nowo

- Obiektowy model analizy
  - Podstawowe klasy występujące w systemie
    - Klasa AtBeggining – Po kliknięciu na ekran ładuje główną scenę gry.
    - Klasa GameManager – Po załadowaniu sceny gry rozpoczyna ją, podczas kolizji z obiektem ładuje scenę końca gry oraz możliwość rozpoczęcia gry na nowo.
    - Klasa AddPoints – Przy przeleceniu między rurami odgrywa dźwięk zdobycia punktu i dodaje punkt.
    - Klasa Points – Na początku gry ustawia punktacje na 0 i aktualizuje punktacje na górze ekranu.
    - Klasa Fly - ustawiamy szybkość naszych podskoków/latania oraz przy każdym z nich odgrywamy odpowiedni dźwięk
    - Klasa PipeSpawner – przy uruchomieniu gry tworzy nowy obiekt rur z losowo umiejscowiona luką między nimi oraz ustawia czas między stworzeniem nowych oraz po jakim czasie zostaną zniszczone.
    - Klasa Movement - Ustawia prędkość poruszania się rur
    - Klasa Quit – Po kliknięciu przycisku wyłącza grę

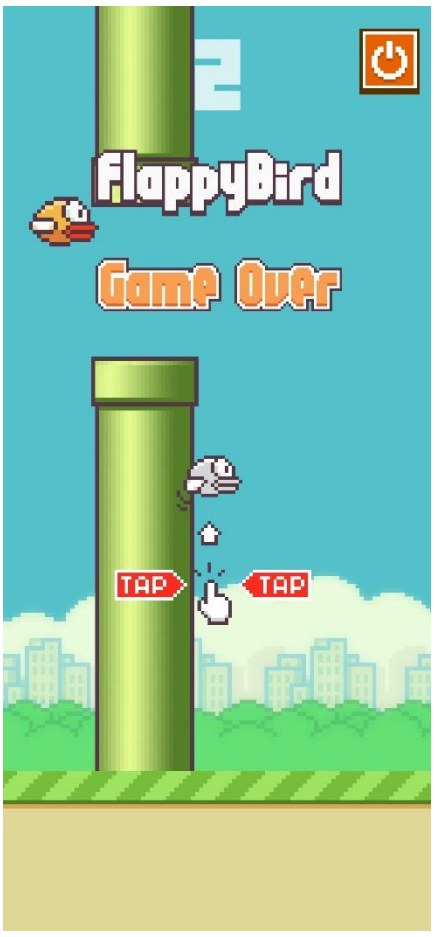
- Projekt interfejsu użytkownika IRS



Rysunek 4



Rysunek 5



Rysunek 6

Interfejs w głównym menu (Rysunek 4) składa się z 2 rzeczy:

- przycisku , który pozwala nam wyłączyć aplikację
- możliwość uruchomienia gry poprzez kliknięcie w dowolne miejsce na ekranie

Interfejs w widoku rozgrywki (Rysunek 5) zawiera:

- możliwość kliknięcia w dowolne miejsce na ekranie by wykonać skok/lot postaci
- widok aktualnych punktów

Interfejs ekranu końca (Rysunek 6) gry:

- możliwość wyjścia z gry
- możliwość rozpoczęciu gry na nowo jak w menu głównym
- wgląd na uzyskane punkty podczas gry

5. Implementacja

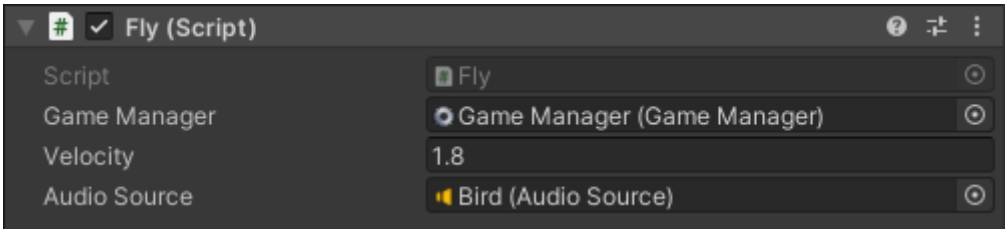
- System poruszania się
  System poruszania się w grze używa tak naprawdę animacji i to główne nie naszej postaci. W przypadku Flappy Bird, gdy się poruszamy to tak naprawdę utrzymujemy naszą postać na poziomie luki w nadchodzących rurach.
  Spadanie naszej postaci pochodzi z dodanego w Unity komponentu **Rigidbody 2D,** a skakanie z klasy **Fly** pokazanej niżej.

```
public GameManager gameManager;
1 reference
public float velocity = 1;
2 references
private Rigidbody2D rb;
[SerializeField]
2 references
private AudioSource _audioSource;
// Start is called before the first frame update
0 references
void Start()
{
    rb = GetComponent<Rigidbody2D>();
    _audioSource = GetComponent<AudioSource>();
}
// Update is called once per frame
0 references
void Update()
{
    if(Input.GetMouseButtonDown(0))
    {
        //Skok
        rb.velocity = Vector2.up * velocity;
        _audioSource.Play();
    }
}

0 references
private void OnCollisionEnter2D(Collision2D collision)
{
    gameManager.GameOver();
}
```

Rys 7

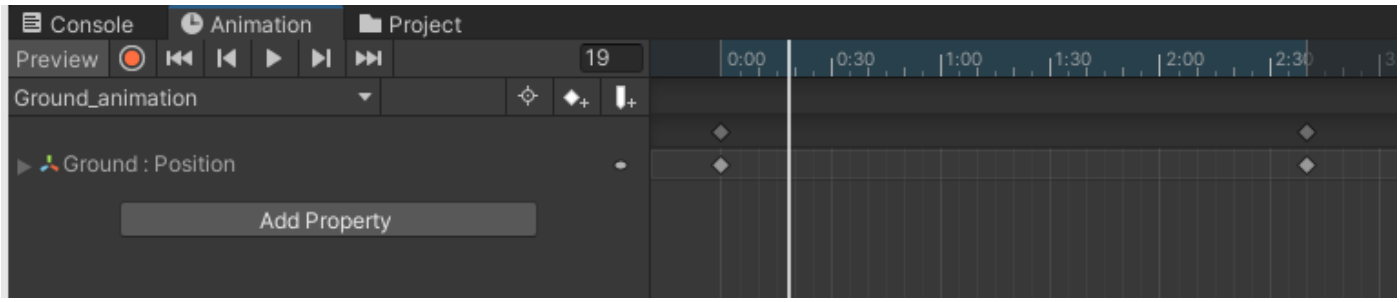
Aby nasza postać latała/skakała ustalamy zmienną zmiennoprzecinkową velocity, która odpowiada jak wysoko będzie nasza postać skakać.

W momencie dotknięcia ekranu nasza postać jest wysyłana z aktualnego miejsca w górę zgodnie z wzorem. W tym samym czasie jest też odtwarzany dźwięk skoku.



Rys 8

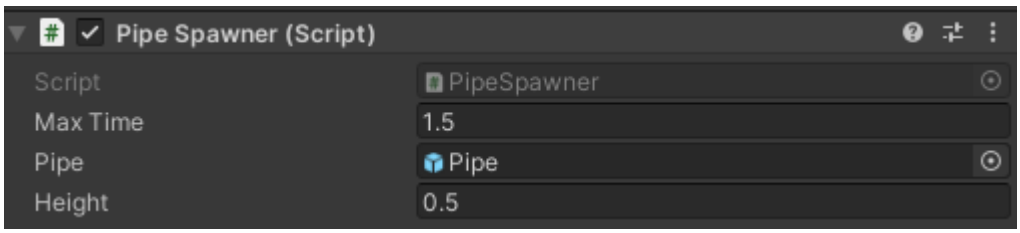
Sama animacja ziemi, która najbardziej daje nam uczucie poruszania się jest zrobiona z pomocą edytora animacji w unity. Po krótce działa on tak ustawiamy element jako pierwszą klatkę, w animatorze dodajemy pierwszy klucz, następnie ustawiamy element w pozycji kończącej i dodajemy drugi. Na końcu wystarczy przesunąć pierwszy klucz na sam początek animacji rozszerzyć długość animacji na osi i ustawić drugi klucz na końcu animacja się wtedy wykona.



Rys 9

Teraz najważniejszy element naszej gry czyli tworzenie rur. Do tego użyliśmy nowego obiektu, którego nazwaliśmy pipe spawner jest to obiekt umieszczony na prawo od ekranu rozrywki po za widokiem gracza na poziomie wysokości startowej naszej postaci. Używa on klasy PipeSpawner i wcześniej przygotowanego obiektu Pipe, który składa się z 2 rur i 3 komponenty Box Collider 2D(1 do zdobycia punktów i 2 do śmierci postaci).

Ustalamy zmienne do timera dzięki, którem będziemy tworzyć rury w równym odstępie czasu. Następnie są one tworzone na losowej wysokości w podanym przez nas przedziale w tym przypadku jest to 0.5. Następnie są one usuwane po upływie 10 sekund.

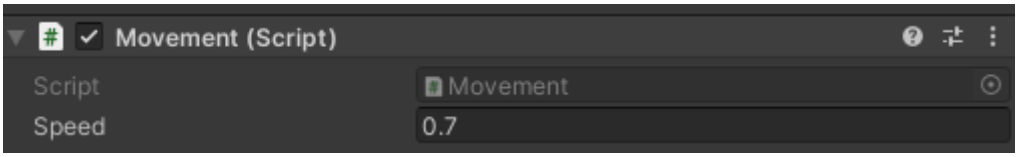


```
1 reference
public float maxTime = 1;
3 references
private float timer = 0;
2 references
public GameObject pipe;
4 references
public float height;

// Start is called before the first frame update
0 references
void Start()
{
    GameObject newpipe = Instantiate(pipe);
    newpipe.transform.position = transform.position + new Vector3(0, Random.Range(-height,height),0);
    Destroy(newpipe, 10f);
}

// Update is called once per frame
0 references
void Update()
{
    if(timer > maxTime)
    {
        GameObject newpipe = Instantiate(pipe);
        newpipe.transform.position = transform.position + new Vector3(0, Random.Range(-height,height),0);
        Destroy(newpipe, 10f);
        timer = 0;
    }

    timer += Time.deltaTime;
}
```

Rys 10

Obiekt Pipe do jego ruchu używamy klasy Movement.

```
public float speed;
// Update is called once per frame
0 references
void Update()
{
    transform.position += Vector3.left * speed * Time.deltaTime;
}
```



Jak też widać w edytorze ustalamy szybkość poruszających się obiektów Pipe.

- Śmierć

Do śmierci w naszej grze dochodzi przy zderzeniu z rurami, podłogą lub próbą wyjścia poza granice gry. Sama śmierć jest dodana poprzez funkcje Fly oraz collidery.

Przy zderzeniu colliderów wywoływana jest funkcja onCollisonEnter2D (Rys 7), która prowadzi do GameManager funkcji GameOver().

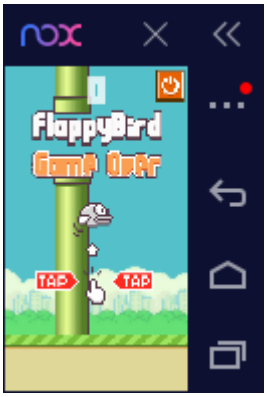
```
public GameObject gameOverCanvas;
[SerializeField]
1 reference
private AudioSource _audioSource;
// Start is called before the first frame update
0 references
private void Start()
{
    Time.timeScale = 1;
}

1 reference
public void GameOver()
{
    _audioSource.Play();
    gameOverCanvas.SetActive(true);
    Time.timeScale = 0;
}

0 references
public void Replay()
{
    SceneManager.LoadScene(1);
}
```

Rys 11

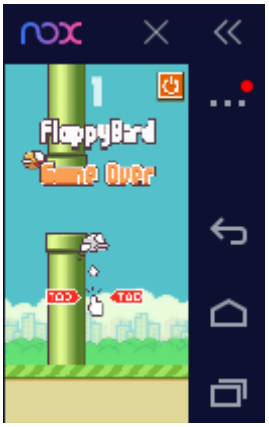
Sam Manager jest też odpowiedzialny za funkcję Replay() służącą do rozpoczęcia na nowo rozgrywki.

6. Testy aplikacji

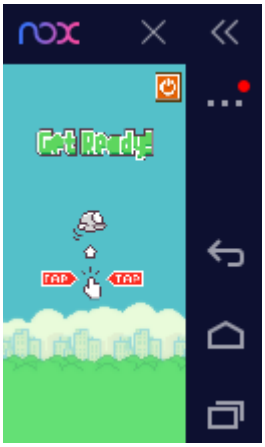
- **Błędy związane z ograniczeniami narzucanymi przez określony OS.**
  Nie zwróciliśmy uwagi na żadne błędy spowodowane ograniczeniami OS. Nasza aplikacja jest na tyle mało złożona, że nie ma żadnego problemu z ładowaniem.
- **Testy na emulatorach i testy na realnych urządzeniach.**
  Grę przetestowaliśmy na telefonie OnePlus 7 Pro na 11 wersji androida zarówno jak i emulatorach. Rozgrywka jest płynna i nie widzimy żadnych problemów.
- **Testy poprawności instalacji i deinstalacji na różnych urządzeniach/emulatorach.**
  Testy na wyżej wymienionym telefonie oraz emulatorze Nox z 7 wersją androida i wyżej nie wskazują na żadne problemy z instalacją lub deinstalacją.
- **Testy związane z ilością wykorzystanej pamięci.**
  Testując zużycie średnie użycie pamięci RAM to około 78MB.
- **Testy GUI**
  Warto wspomnieć, że wymusiliśmy by aplikacja blokowała się w trybie portretowym.
  Aspect ratio:



16 : 9 Portrait



18 : 9 Portrait



19 : 9 Portrait

Jak też widać przy bardziej pospolitych wielkościach telefonu skalowanie nie ma większego problemu.

7. Tabela Oceny

Tabela oceny projektu z Programowania urządzeń mobilnych			
Ip	Oceniany element	max	punkty
1	Ogólny opis gry	1	
2	Analiza dziedziny	2	
3	Specyfikacja wymagań	2	
4	Analiza i projekt:		
	Architektura systemu gry	2	
	Obiektowy model analizy	2	
	Projekt oprogramowania	2	
5	Czytelność kodu źródłowego	2	
6	Podział kodu na moduły/biblioteki/klasy	1	
7	Zgodność gry z oryginałem	6	
8	Dźwięki	4	
9	Muzyka	1	
10	Sterowanie/nawigacja	2	
11	Dodatkowa wersja z rozbudowaną grafiką, dźwiękiem, muzyką	5	
12	Inwencja własna	3	
13	Punkty przyznane od prowadzącego	3	
		40	

GitHub