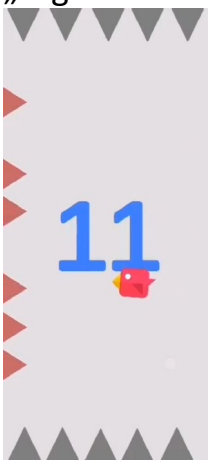
DONT TOUCH THE SPIKES

Autor: Paweł Pauszek

Spis treści

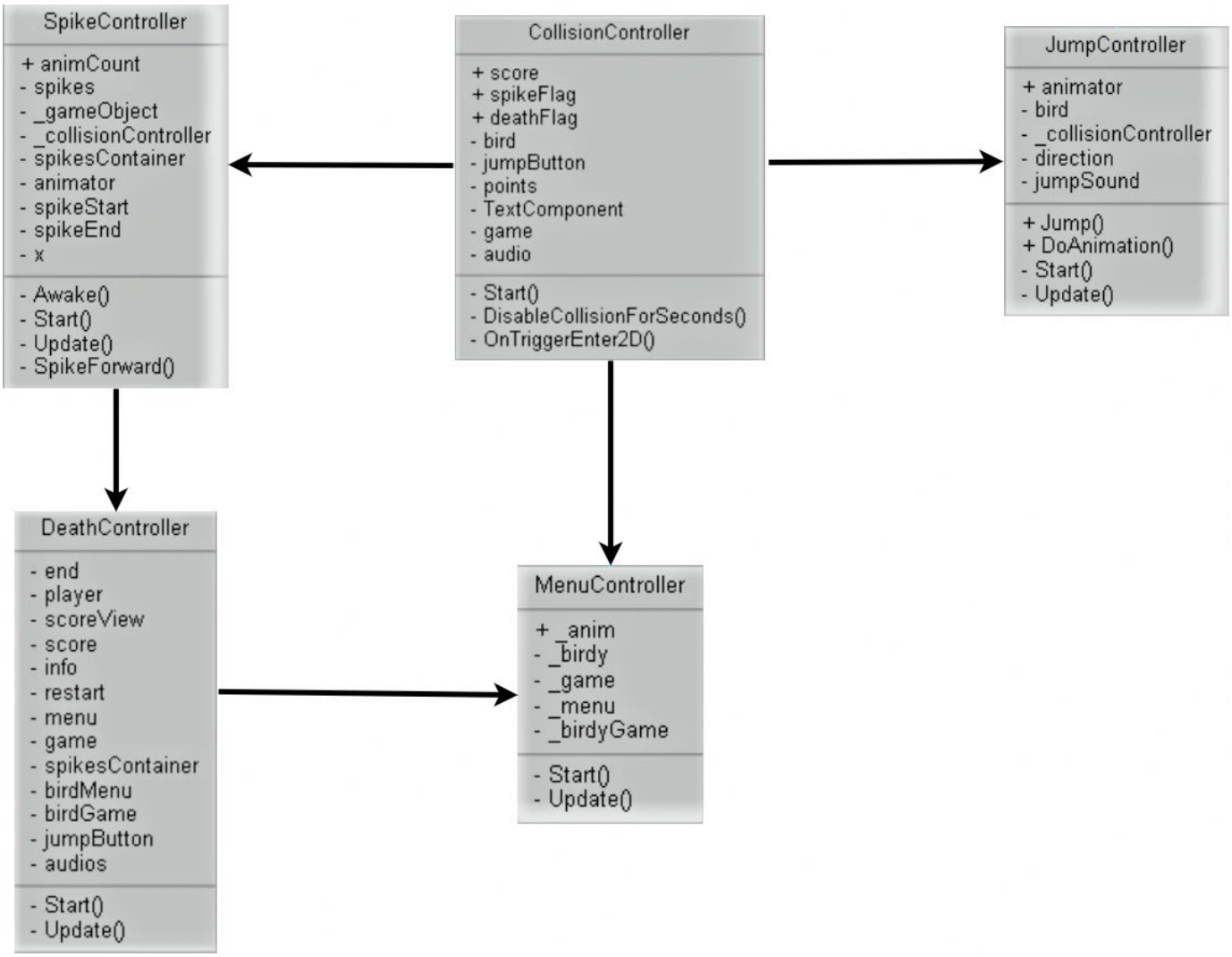
I. Ogólny opis gry - koncepcja

Celem projektu były stworzenie gry na wzór Don’t Touch The Spikes.
Gra opiera się na odbijaniu się ptakiem od ścian, jednocześnie unikając kolców na ścianie.
Jedyną a zarazem główna postacią w grze jest wyżej wymieniony ptak, którym sterujemy.
Granicą systemu są 4 otaczające gracza ściany. Dwie z nich (górna i dolna) zawierają wyłącznie kolce, bez miejsc wolnych i możliwości odbicia się od ściany. Użytkownik po dotknięciu ich od razu „umiera”.
Sterowanie polega na naciśnięciu w odpowiednim momencie z odpowiednią ilością razy dowolnego miejsca na ekranie, skutkuje to wybiciem się ptaka w wzwyż oraz w kierunku ściany.
Celem użytkownika jest zdobycie jak największej ilości punktów odbijając się od ścian.
Po dotknięciu ściany (lewej lub prawej) następuje odbicie się ptaka od niej i zmiany kierunku ruchu na przeciwny, tak aby leciał w stronę drugiej ściany oraz zostaje dodany punkt.
Kolce znajdujące się na ścianie lewej lub prawej, są generowane w sposób losowy. Generowanie kolców to proces dwu etapowy: wpierw jest losowana ilość wolnych miejsc (od 1 aż do 3) a następnie następuje losowanie w zależności od wolnych miejsc numerów kolców które są dezaktywowane.
Ilość przedziału losowania wolnych miejsc na starcie gry wynosi stałe 3/3 i zmniejsza się o 1 co 20 punktów aż do 1 co w rezultacie przy wyniku ponad 40 daje 33% procent szansy na każdą z wolnych miejsc.
Wraz z poziomem trudności zmienia się także muzyka. Na starcie dość spokojna a z każdym poziomem trudności coraz bardziej dynamiczna.
Po śmierci gracza włącza się „smutna muzyka” i zostaje wyświetlony aktualnie zdobyty wynik a pod nim aktualnie najwyższy wcześniej zdobyty wynik, tzw: „High score”



II. Analiza dziedziny

Klasy w dziedzinie gry oraz ich diagram klas wraz z relacjami



Opis atrybutów:

MenuController:

Atrybut	Opis
_birdy	obiekt ptaka w menu głównym
_anim	animator do obiektu birdy (statycznie przydzielona animacja)
_game	canvas funkcjonalnej gry

_menu	canvas menu
_birdyGame	obiekt właściwego ptaka, którym skacze gracz

JumpController:

Atrybut	**Opis**
bird	obiekt skaczącego ptaka w grze
animator	animator do obiektu bird
_collisionController	obiekt pomocniczy, służący do zczytania danych z klasy CollisionController
direction	zmienna pozwalająca określić kierunek skoku ptaka
jumpSound	służy do odtwarzania dźwięku przy skoku
metoda Jump()	ptak leci w górę i lekko w kierunku direction
metoda DoAnimation()	ruch „skrzydłami” przy skoku

CollisionControleler:

Atrybut	**Opis**
score	określa aktualny wynik gracza
spikeFlag	atrybut służący do określania czy nastąpiła kolizja, by wysunąć kolce
deathFlag	określa czy gracz „umarł”
bird	obiekt ptaka gracza
jumpButton	przycisk skoku (dowolny punkt na ekranie)
points	obiekt odwołujący się do TextComponent, wyświetlający aktualny score
game	canvas gry
audio	obsługa zmiennej, dynamicznej muzyki w tle
metoda DisableCollisionForSeconds()	przy jakiejkolwiek kolizji ptaka, następuje wyłączenie dla niego

	Collidera na 0.2 sekundy. Zapobiega to bugowi, który obracał ptaka w nieskończoność
metoda OnTriggerEnter2D()	obsługa zdarzenia po kolizji

SpikeController:

Atrybut	**Opis**
animCount	pomocniczy atrybut do obliczenia ile razy ptak odbił się od ściany
spikes	obiekty kolców
gameObject wraz z animatorem	obiekt ptaka gracza
spikesContainer	kontener zawierający w sobie wszystkie kolce
spikeStart oraz spikeEnd	pozwala obliczyć od którego do którego kolca rozpocząć losowanie dotyczące który kolec zostanie wyłączony
x	poziom trudności (wartość 1-3)
zmienia się samoistnie z przyrostem punktów	
metoda SpikeForward()	wysuwa kolce z lewej lub prawej strony

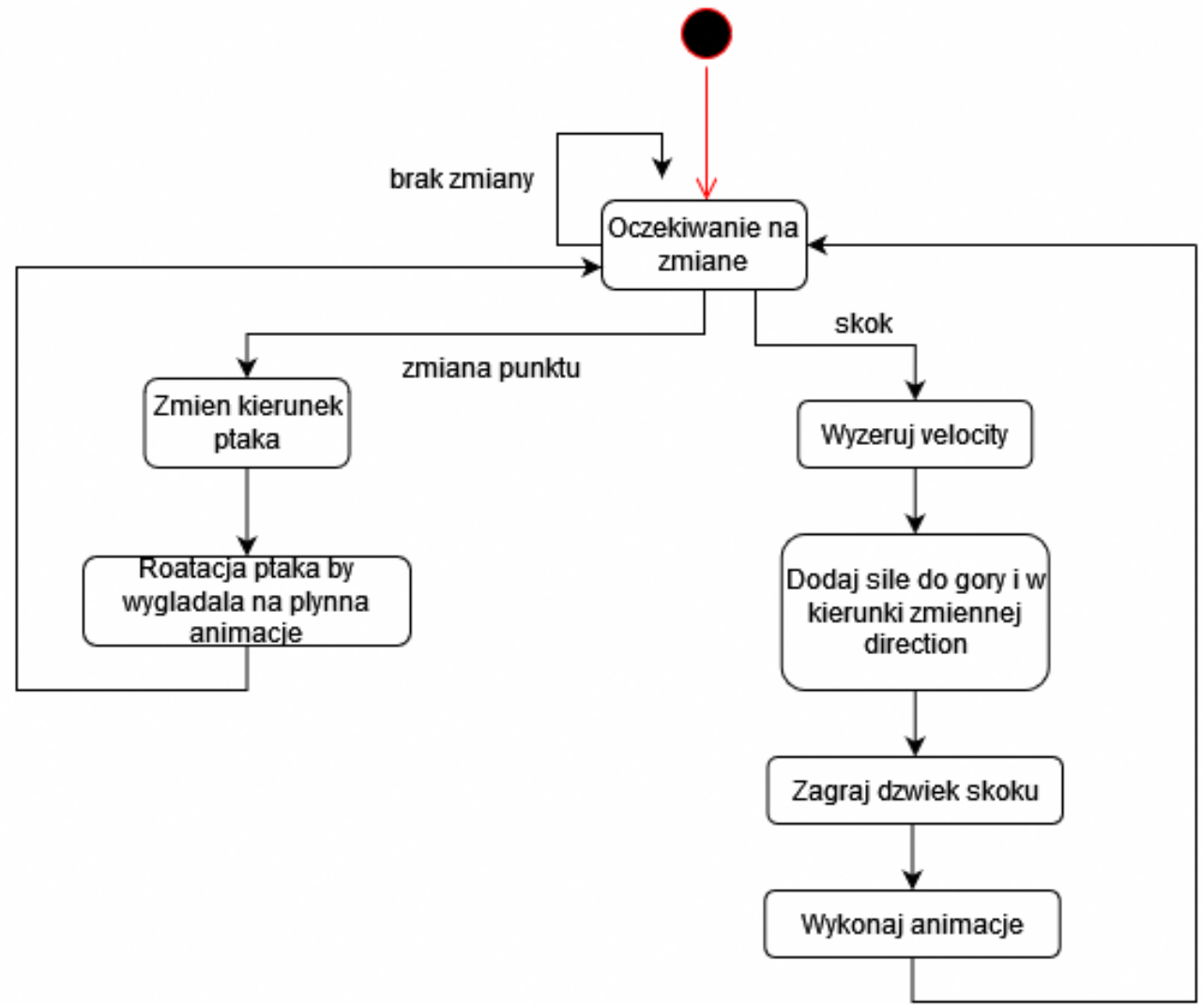
DeathController:

Atrybut	Opis
end	canvas „śmierci”, włącza się po śmierci gracza. Prezentuje uzyskany wynik oraz highscore
player	obiekt ptaka
scoreView	TextMeshPro do wyświetlenia wyniku
score	uzyskany wynik
info /restart	informacja o restarcie gry na nowo (po śmierci możemy na nowo

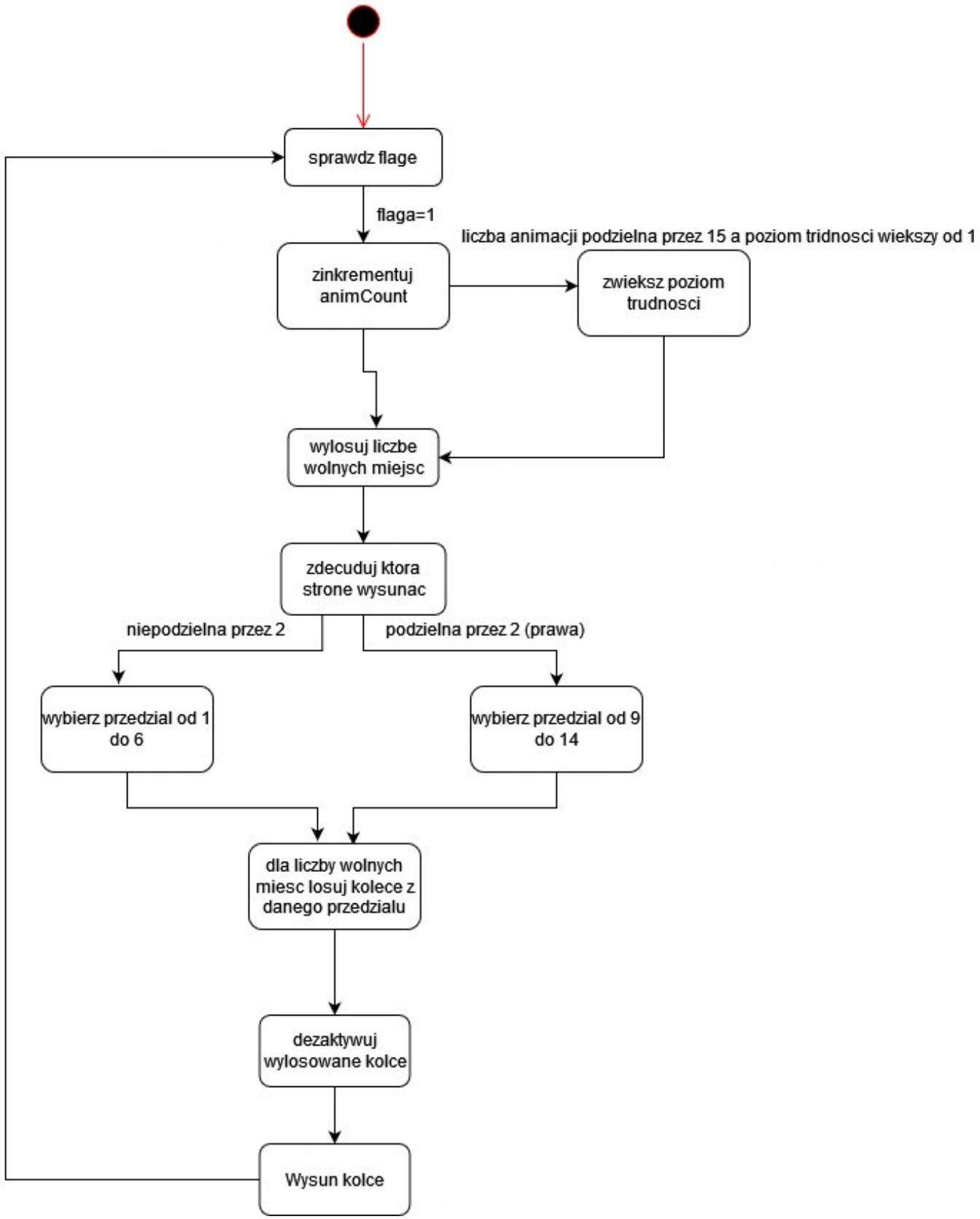
	rozpocząć grę, bez wychodzenia z aplikacji)
menu	odwołanie do canvas menu
game	odwołanie do canvas gry
spikesContainer	kontener na kolce
birdMenu	ptak z menu (zapętlona animacja), włącza się po restarcie gry
birdGame	ptak z gry (leżący na dolnych kolcach po śmierci)
jumpButton	wyłączenie możliwości skakania po śmierci
audios	dostęp do obiektów muzyki w celu ich zmiany

Diagram stanów dla klas:

Skoku

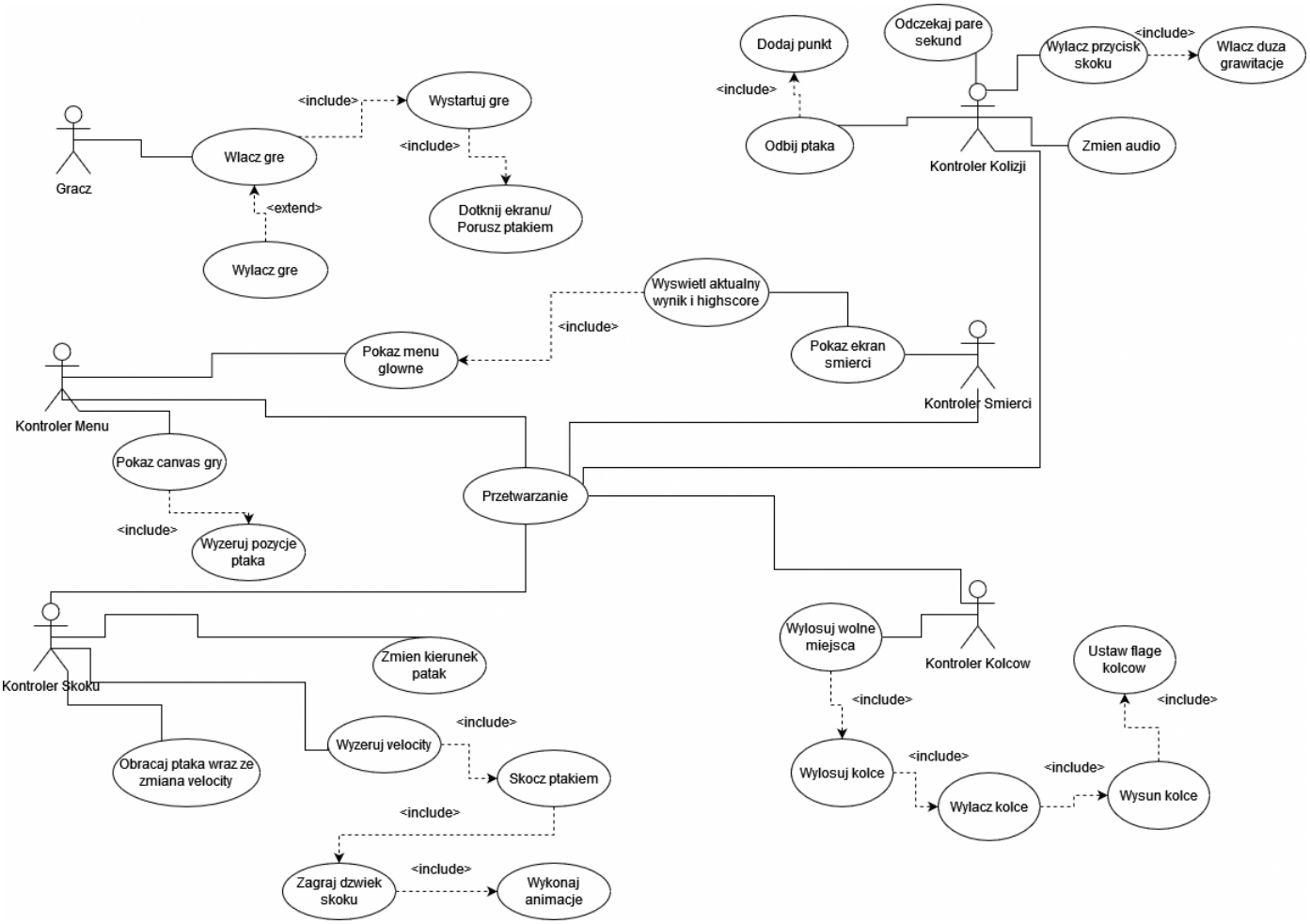

brak zmiany
Oczekiwanie na zmiane
skok
zmiana punktu
Zmien kierunek ptaka
Roatacja ptaka by wygladala na plynna animacje
Wyzeruj velocity
Dodaj sile do gory i w kierunki zmiennej direction
Zagraj dzwiek skoku
Wykonaj animacje

Kolców


sprawdz flage
flaga=1
liczba animacji podzielna przez 15 a poziom tridnosci wiekszy od 1
zinkrementuj animCount
zwieksz poziom trudnosci
wylosuj liczbe wolnych miejsc
zdecuduj ktora strone wysunac
niepodzielna przez 2
podzielna przez 2 (prawa)
wybierz przedzial od 1 do 6
wybierz przedzial od 9 do 14
dla liczby wolnych miesc losuj kolece z danego przedzialu
dezaktywuj wylosowane kolce
Wysun kolce

III. Specyfikacja wymagań

Ogólny diagram przypadków użycia:



Definicje przypadków użycia:

Nazwa	Skocz ptakiem
Aktorzy	Kontroler Skoku, Gracz, Kontroler Menu, Kontroler Śmierci
Warunki wstępne	Canvas gry jest aktywny, Gracz dotknął ekranu, Ptak nie jest martwy
Warunki końcowe	Ptak poleciał do góry
Scenariusz	1. Wczytaj kierunek ptaka

	2. Dodaj sile do obiektu ptaka (x:1.5*kierunek, y:2)

Nazwa	Wylosuj wolne miejsca
Aktorzy	Kontroler Kolców, Kontroler Kolizji
Warunki wstępne	Ptak odbił się od ściany
Warunki końcowe	Wylosowanie ilości miejsc
Scenariusz	1. Sczytaj poziom trudności 2. Sczytaj liczbę odbić ptaka od ściany 3. Zmiana poziomu trudności gdy wynik jest dostatecznie wysoki 4. Wylosowanie liczby wolnych miejsc z przedziału 1-3 w zależności od poziomu trudności

Nazwa	Wyświetl aktualny wynik i highscore
Aktorzy	Kontroler Śmierci
Warunki wstępne	Gracz umarł, Ekran śmierci jest aktywny
Warunki końcowe	Wyświetlenie wyników
Scenariusz	1. Wczytanie aktualnego wyniku 2. Porównanie aktualnego wyniku z najwyższym highscore 2a. Ewentualne ustawienie aktualnego wyniku jako highscore 3. Wyświetlenie wyników

IV. Analiza i projekt

Do projektu podszedłem trochę inaczej niż tradycyjni programiści. Zamiast używania schematu MVC lub innych popularnych tego typu wzorców, postanowiłem z racji na prostotę projektu oraz wygodę wykorzystać tylko i wyłącznie kontrolery. To w nich się wszystkie dzieje i to one wszystkim zarządzają. Z racji iż projekt pisałem na silniku Unity, ważne rzeczy deklarowałem nie w kodzie a w samej aplikacji. Tym sposobem w kodzie odwoływałem się do obiektów, modyfikując je lub po prostu wczytując ich wartości.

Zamiast korzystać ze scen by zmieniać widok, posłużyłem się po prostu Canvasem z zależności Unity. Więc w przypadku zmiany jakiegokolwiek widoku, nakładem na pierwszy plan dany Canvas zamiast ładowania scen.

Starając się przyrównać mój schemat gry do modelu MVC wyglądałoby to tak:

Model:

- Obiekt ptaka gry
- Obiekt ptaka menu
- Kolce
- Ściany

View:

- Widok główny (Menu)
- Widok gry
- Widok śmierci

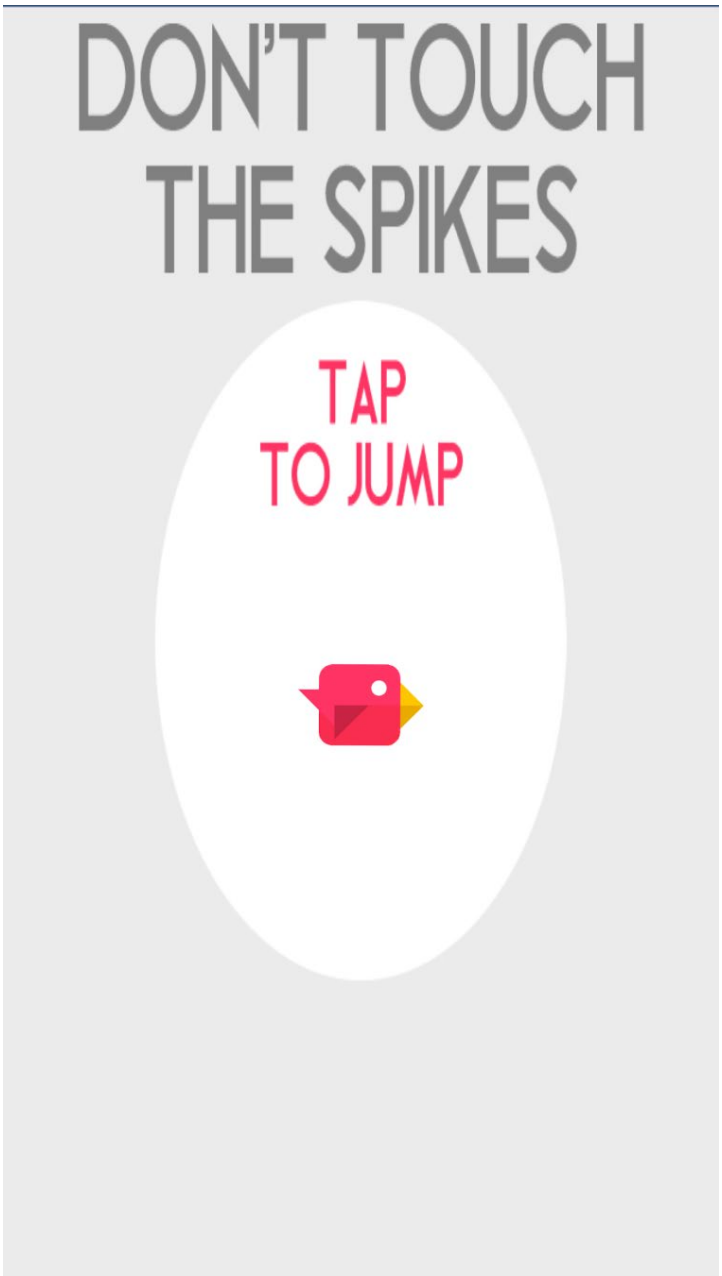
Controller:

- DeathController
- JumpController
- CollisionController
- MenuController
- SpikeController

Lista opisów, atrybutów oraz metod dla kontrolerów została przedstawiona powyżej, w rozdziale 2.

Projekt interfejsu użytkownika IRS

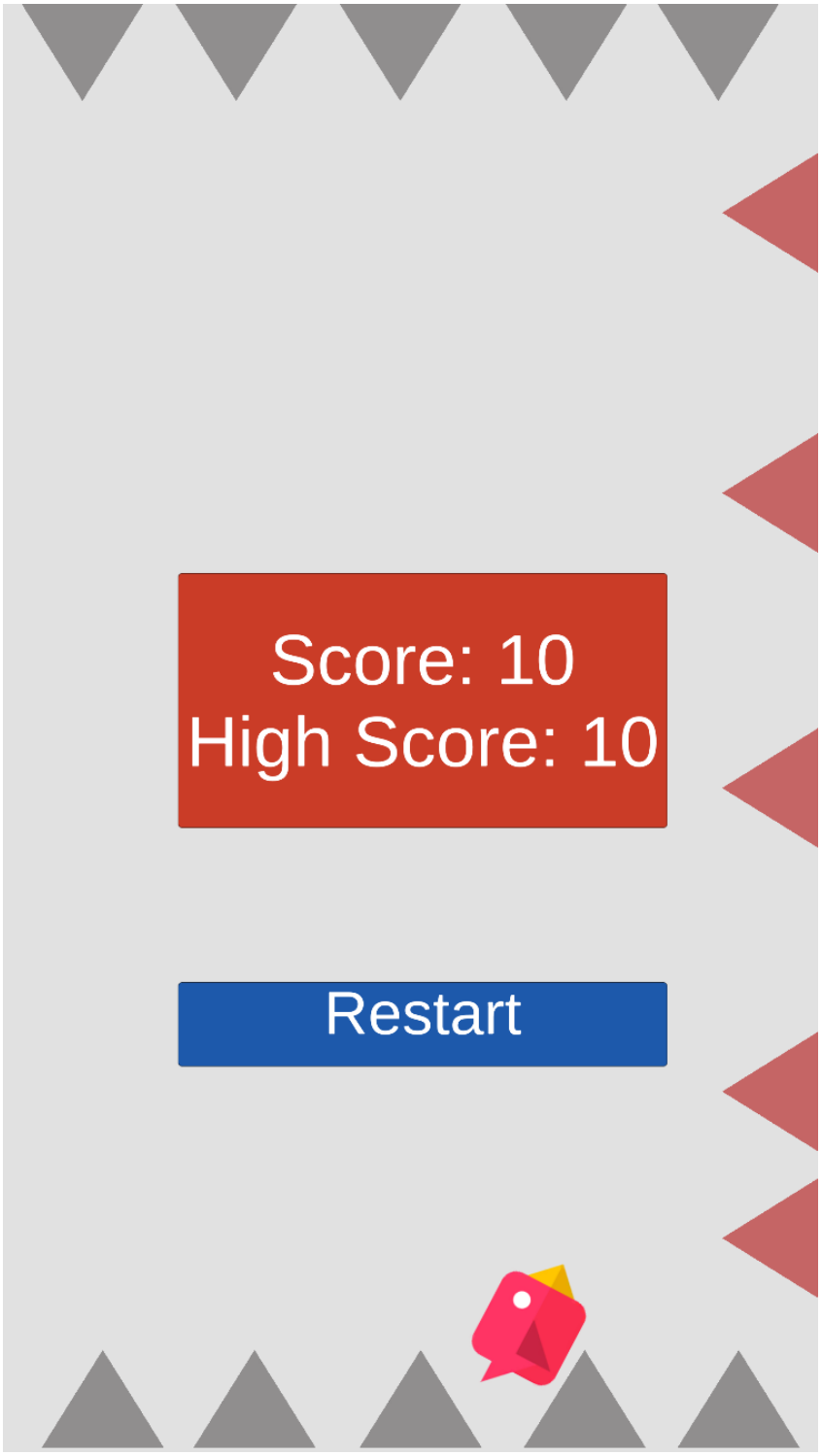
Interfejs menu głównego:



Po naciśnięciu gdziekolwiek na ekran, gra się rozpocznie.

Canvas gry:

Ekran śmierci:



Ekran pośmiertny wyświetla zdobyty aktualnie wynik oraz największy kiedykolwiek zdobyty. Wystarczy kliknąć gdziekolwiek by przejść do menu głównego.

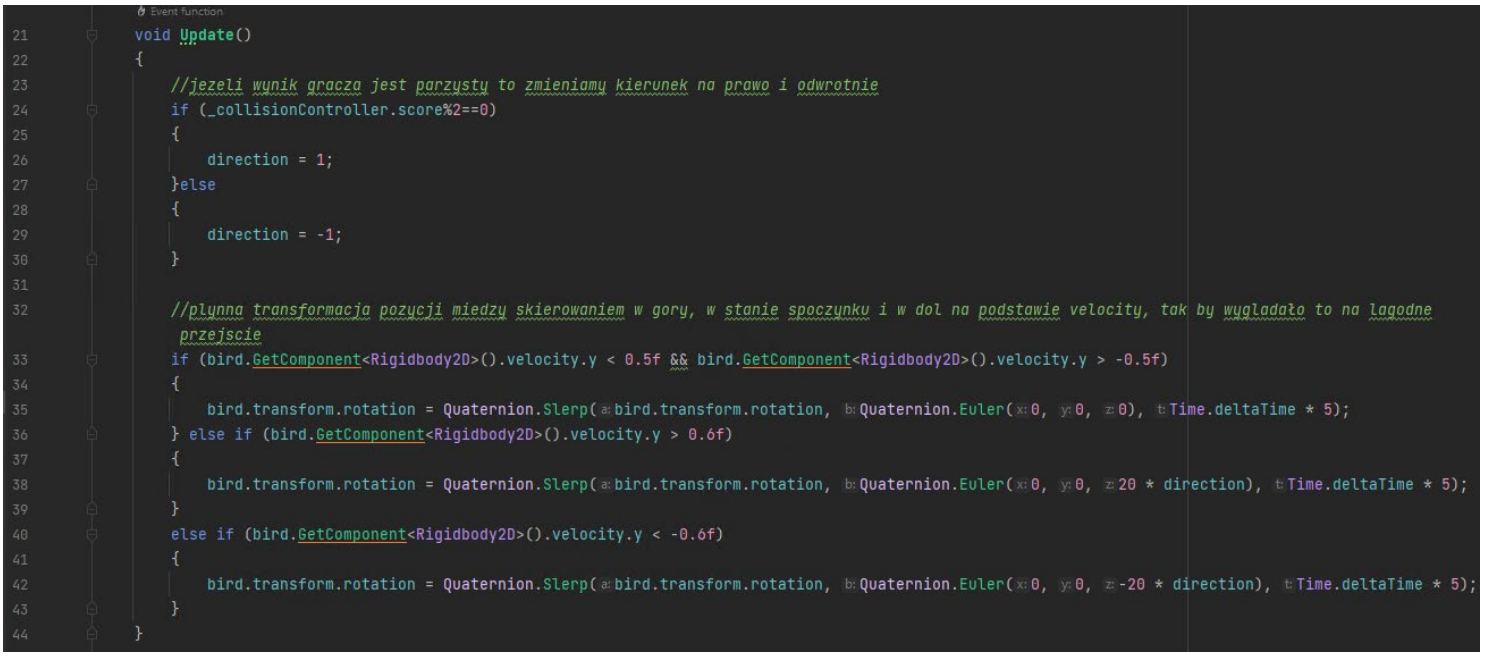
V. Implementacja

Z racji nawyku opisywania kodu w źródle, wklejam wyłącznie zdjęcia kodów. Pozwoliłem sobie pominąć wczytywanie obiektów oraz ich implementacje w funkcji Start()

KontrolerMenu:

```
26
        Event function
27      void Update()
28      {
29          if (Input.GetButtonDown("Fire1"))
30          {
31              //rozpoczecie gry
32              Vector3 birdyPos = _birdy.transform.position;
33              _anim.SetLayerWeight( layerIndex: 1, weight: 0);  //wylaaczanie animacji menu
34              _birdyGame.transform.position = birdyPos;
35              _birdyGame.GetComponent<Rigidbody2D>().gravityScale = 0.4f; //wlaczenie grawitacji ptaka
36              _menu.gameObject.SetActive(false); //wylaczenie menu
37              _game.gameObject.SetActive(true); //wlaczenie canvasu gry
38              _birdyGame.SetActive(true); //wlaczenie ptaka do gry
39              _birdyGame.transform.position = new Vector3( x: 0, y: 0, z: 0); //wyzerowanie pozycji ptaka
40              _birdyGame.transform.localScale = new Vector3( x: 0.5f, y: 0.5f, z: 1); //ustawienie kierunku ptaka
41              _birdy.gameObject.SetActive(false); //wylaczenie ptaka z menu
42              _birdyGame.GetComponent<PolygonCollider2D>().enabled = true;
43          }
44      }
45  }
46
```

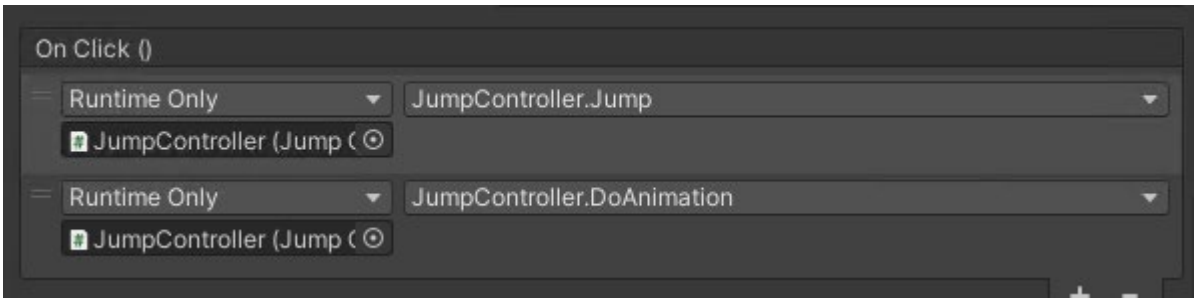
KontrolerSkoku:



```
    Event function
21  void Update()
22  {
23      //jezeli wynik gracza jest parzysty to zmieniamy kierunek na prawo i odwrotnie
24      if (_collisionController.score%2==0)
25      {
26          direction = 1;
27      }else
28      {
29          direction = -1;
30      }
31
32      //plynna transformacja pozycji miedzy skierowaniem w gory, w stanie spoczynku i w dol na podstawie velocity, tak by wygladało to na lagodne
         przejscie
33      if (bird.GetComponent<Rigidbody2D>().velocity.y < 0.5f && bird.GetComponent<Rigidbody2D>().velocity.y > -0.5f)
34      {
35          bird.transform.rotation = Quaternion.Slerp( a: bird.transform.rotation, b: Quaternion.Euler( x: 0,  y: 0,  z: 0), t: Time.deltaTime * 5);
36      } else if (bird.GetComponent<Rigidbody2D>().velocity.y > 0.6f)
37      {
38          bird.transform.rotation = Quaternion.Slerp( a: bird.transform.rotation, b: Quaternion.Euler( x: 0,  y: 0,  z: 20 * direction), t: Time.deltaTime * 5);
39      }
40      else if (bird.GetComponent<Rigidbody2D>().velocity.y < -0.6f)
41      {
42          bird.transform.rotation = Quaternion.Slerp( a: bird.transform.rotation, b: Quaternion.Euler( x: 0,  y: 0,  z: -20 * direction), t: Time.deltaTime * 5);
43      }
44  }
```

```
45
        1 asset usage
46      public void Jump(){
47          //na klik, zrestuj sily na ptaka oraz dodaj nowa sile w gore i lewo lub prawo
48          bird.GetComponent<Rigidbody2D>().velocity = Vector2.zero;
49          bird.GetComponent<Rigidbody2D>().AddForce(new Vector2( x: 1.5f*direction,  y: 2f), ForceMode2D.Impulse);
50          jumpSound.Play();
51          bird.transform.localScale = new Vector3( x: 0.5f*direction,  y: 0.5f,  z: 1);
52      }
53
        1 asset usage
54      public void DoAnimation(){
55          //na klik wykonaj animacje latania
56          animator.Play( stateName: "flyingAnimation", layer: 0, normalizedTime: 0.0f);
57      }
58  }
```

Dodałem przycisk na całą wielkość ekranu. Naciśniecie go powoduje wywołanie funkcji Jump() oraz DoAnimation() .



Kontroler Kolizji:

Funckcja wyłączająca na 0.2 sekundy collider ptaka po tym jak dotknął czegokolwiek, by usunąć buga, który powodował niekończący się obrót przy ścianie i kolcu.

```
28      IEnumerator DisableCollisionForSeconds(float seconds)
29      {
30          bird.GetComponent<PolygonCollider2D>().enabled = false;
31          yield return new WaitForSeconds(seconds);
32          bird.GetComponent<PolygonCollider2D>().enabled = true;
33      }
34
```

```
35      void OnTriggerEnter2D(Collider2D coll)
36      {
37          if (!coll.gameObject.CompareTag("UpperWall") && !coll.gameObject.CompareTag("DownWall"))
38          {
39              StartCoroutine( routine: DisableCollisionForSeconds(0.2f));
40          }
41
42          Vector2 force = bird.GetComponent<Rigidbody2D>().velocity;
43          bird.GetComponent<Rigidbody2D>().velocity = Vector2.zero;
44          bird.GetComponent<Rigidbody2D>().AddForce(new Vector2(-force.x,  y: 1f), ForceMode2D.Impulse);
45          //w przypadku kolizji z przeszkoda gora lub dol lub spike
46          if (coll.gameObject.CompareTag("UpperWall") || coll.gameObject.CompareTag("DownWall")  || coll.gameObject.CompareTag("Spike"))
47          {
48              //wylacz przycisk skoku, wlacz duza grawitacje by ptak szybko spadl
49              jumpButton.SetActive(false);
50              bird.GetComponent<Rigidbody2D>().gravityScale = 3;
51              bird.transform.Rotate( xAngle: 0,  yAngle: 0,  zAngle: 90);
52              if (coll.gameObject.CompareTag("DownWall"))
53              {
54                  //zatrzymaj ptaka na ziemii
55                  bird.GetComponent<Rigidbody2D>().velocity = Vector2.zero;
56                  bird.GetComponent<Rigidbody2D>().gravityScale = 0;
57                  bird.transform.Rotate( xAngle: 0,  yAngle: 0,  zAngle: 30);
58              }
59
60              foreach (AudioSource a in audio)
61              {
62                  a.Stop();
63              }
64
65              deathFlag = 1; //ustaw flage ze ptak umarl
66              TextComponent.text= "";
67          }
```

```
68          else
69          {
70              //w przypadku kolizji ze sciana lewa lub prawo, ustaw przeciwny kierunek zwortu do aktualnego
71              Vector3 scale = bird.transform.localScale;
72              bird.transform.localScale = new Vector3(-scale.x, scale.y, scale.z);
73              //dodaj punkt jezeli ptak jest zywy
74              if (deathFlag==0)
75              {
76                  score++;
77
78                  if (score>11 && audio[1].isPlaying)
79                  {
80                      audio[1].Stop();
81                      audio[2].Play();
82                  }
83
84
85                  if (score>5 && audio[0].isPlaying)
86                  {
87                      audio[0].Stop();
88                      audio[1].Play();
89                  }
90              }
91              //wyswietl punkty
92              TextComponent.text = score.ToString();
93              spikeFlag = 1;
94          }
95      }
96  }
```

Powyższa funkcja wykonuje się przy gdy ptak dotknie czegokolwiek.

Kontroler kolców:

```
32      void Update()
33      {
34          //jezeli nastapila kolizja, wylosuj wolne miejsce z kolcow i przestaw animacje
35          if (_collisionController.spikeFlag==1)
36          {
37              SpikeForward();
38              _collisionController.spikeFlag = 0;
39          }
40      }
```

```
42      void SpikeForward()
43      {
44          //losowanie liczb 1-3, ktore oznaczaja ilosc wolne miejsca
45          animCount++;
46
47          if (animCount%6==0 && x>1)
48          {
49              x--;
50          }
51
52          int freeSpace = Random.Range(x, 3);
53          ArrayList randomSpikeList = new ArrayList();
54          //okreslenie czy zmiana dotyczy lewej czy prawej strony (0-7) to lewa strona, (8-15) to prawa strona
55          if (animCount%2==0)
56          {
57              spikeStart = 9;
58              spikeEnd = 14;
59          }
60          else
61          {
62              spikeStart = 1;
63              spikeEnd = 6;
64          }
65
66          //dla kazdego wolnego miejsca losujemy czy ma byc to 1 wolne czy 2 wolne miejsca
67          for (int i = 0; i < freeSpace; i++)
68          {
69              int randomSpike = Random.Range(spikeStart, spikeEnd);
70              int freeSpikes = Random.Range(0, 1);
71              int spikeTwo = randomSpike + freeSpikes;
72              randomSpikeList.Add(randomSpike);
73              randomSpikeList.Add(spikeTwo);
74          }
```

```
            //te spike ktore zostaly wylosowane jako wolne sa wylaczane
            for (int i = 0; i < spikes.Length; i++)
            {
                if (randomSpikeList.Contains(i))
                {
                    spikes[i].SetActive(false);
                }
                else
                {
                    spikes[i].SetActive(true);
                }
            }

            //granie animacji dla lewej lub prawej strony
            if (animCount==1)
            {
                animator.Play( stateName: "SpikeGo");
            }
            else if (animCount%2!=0)
            {
                animator.Play( stateName: "MoveRight");
            }
            else
            {
                animator.Play( stateName: "MoveLeft");
            }

        }
    }
```

Kontroler śmierci:

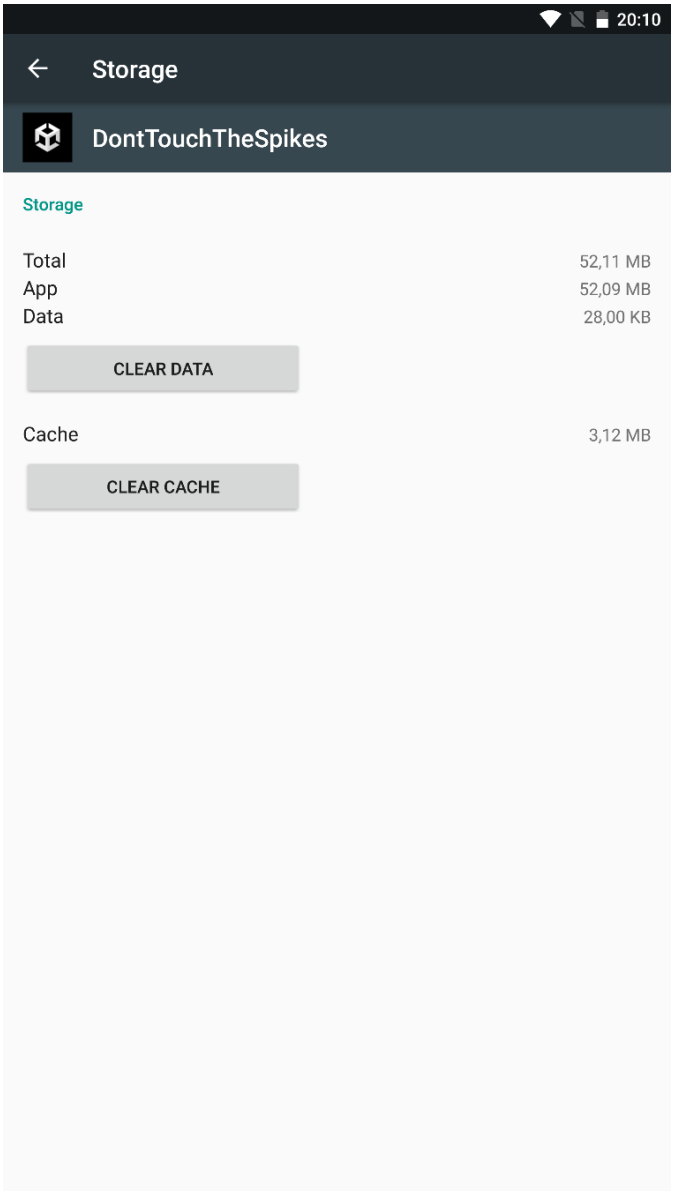
```
    Event function
    void Update()
    {
        if (player.GetComponent<CollisionController>().deathFlag == 1)
        {
            end.SetActive(true);
            //wyswietlenie wyniku oraz highscore nawet przy restrarcie gry
            if (player.GetComponent<CollisionController>().score > PlayerPrefs.GetInt( key: "HighScore",  defaultValue: 0))
            {
                PlayerPrefs.SetInt("HighScore", player.GetComponent<CollisionController>().score);
            }
            score.text = "Score: " + player.GetComponent<CollisionController>().score + "\nHigh Score: " + PlayerPrefs.GetInt( key: "HighScore",
             defaultValue: 0);

            if (Input.GetButtonDown("Fire1"))
            {
                //zrestartowanie gry oraz kluczowych zmiennych
                player.GetComponent<CollisionController>().deathFlag = 0;
                player.GetComponent<CollisionController>().score = 0;
                spikesContainer.GetComponent<SpikeController>().animCount = 0;
                end.SetActive(false);
                birdGame.SetActive(false);
                game.SetActive(false);
                menu.SetActive(true);
                birdMenu.SetActive(true);
                jumpButton.SetActive(true);
                menu.GetComponent<MenuController>()._anim.SetLayerWeight( layerIndex: 1,  weight: 1);
            }
        }
    }
}
```

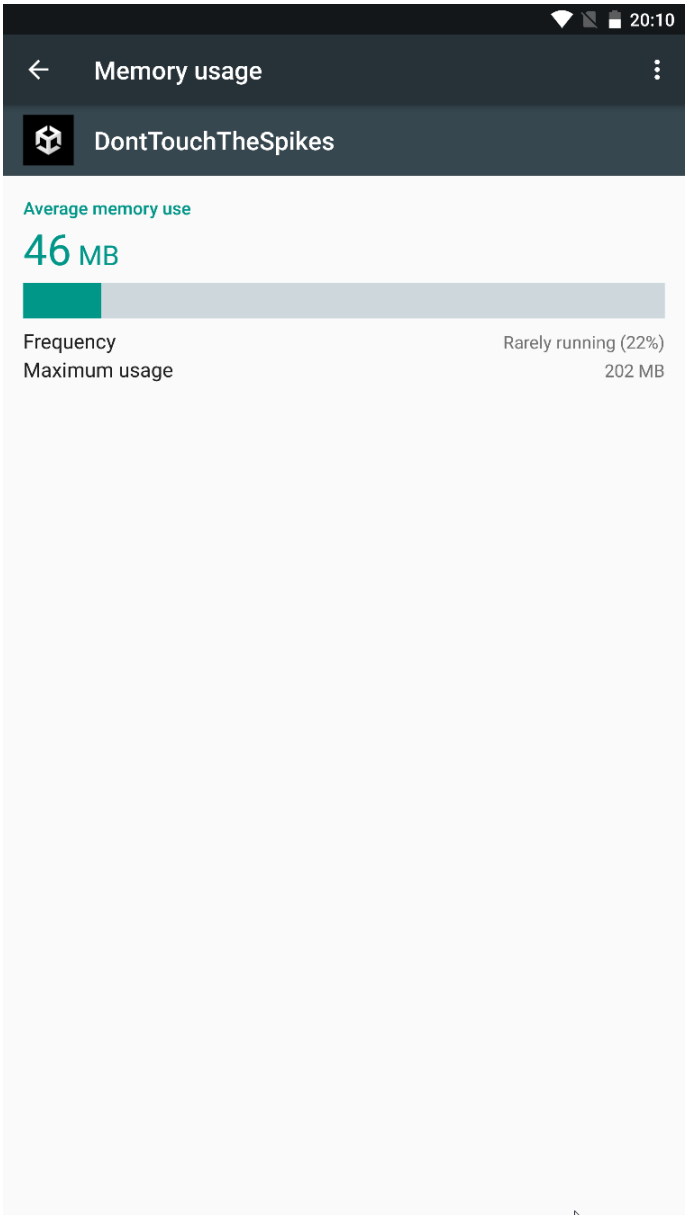
VI. Testy aplikacji

Aplikacja była testowana zarówno na emulatorze NOX jak i na zwykłych telefonach z systemem Android. Instalacja jak i deinstalacja nie sprawiała żadnych problemów. Nie natknąłem się także na błędy związane z ograniczeniami narzuconych przez OS. Aplikacja także poprawnie się skaluje dla różnych rozdzielczości ekranów (nie dotyczy poziomego ustawienia)

Testowanie zużycia pamięci na dysku

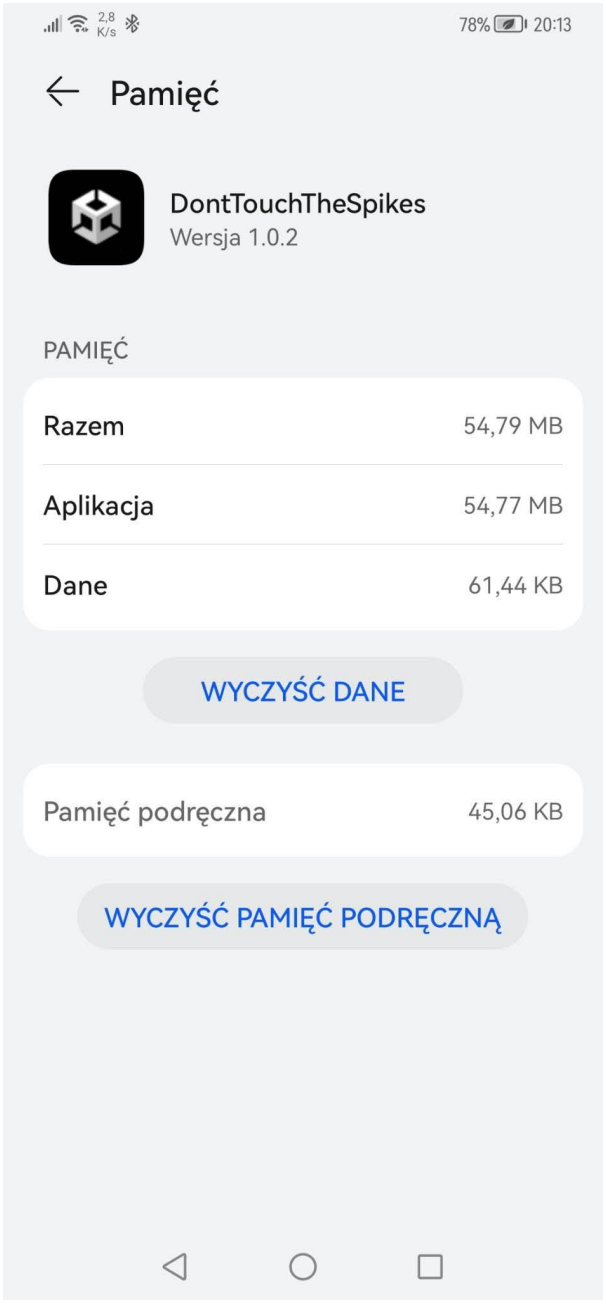


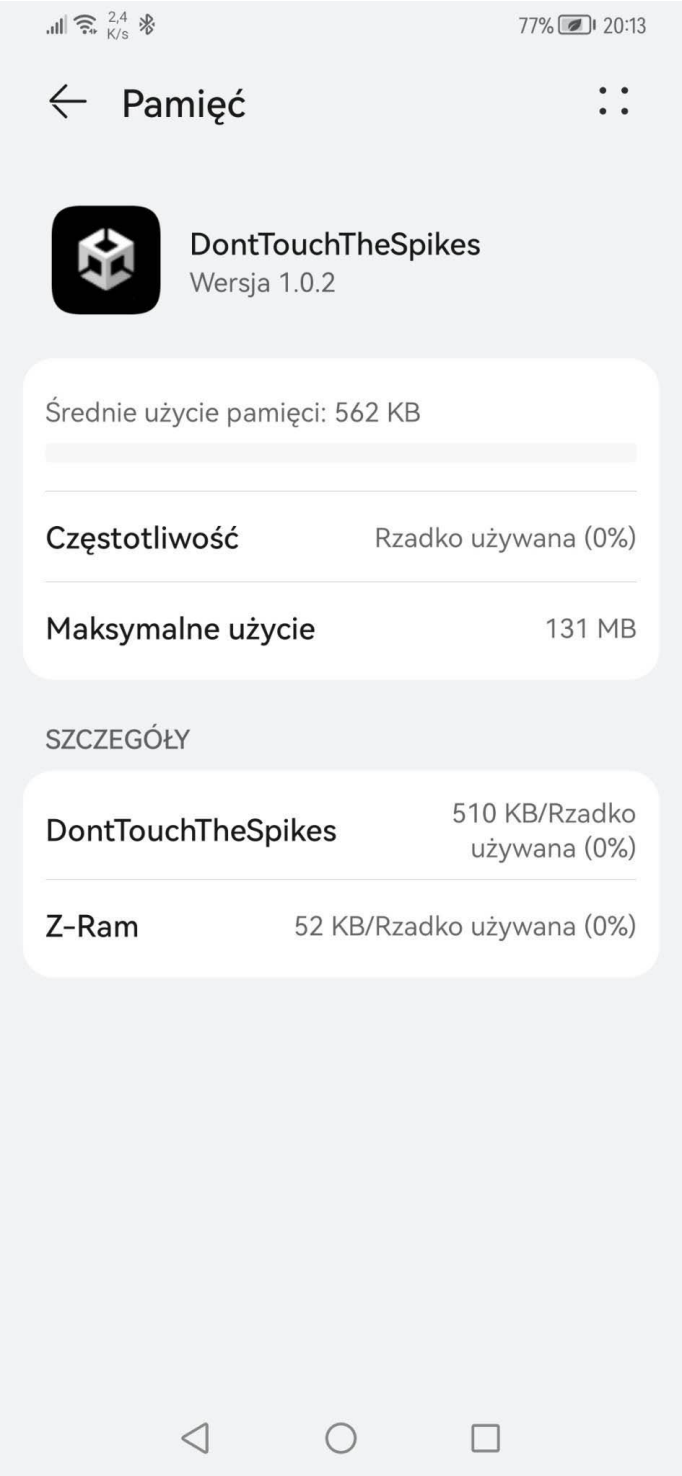
Zużycie pamięci ulotnej:



Jak można zauważyć, w obu przypadkach zajmowana jest bardzo mała przestrzeń.

Dla porównanie poniżej zamieszczam zużycie z telefony Huawei Mate 20 Pro





Niestety nie udało mi się dokonać pomiaru zużycia baterii ani na emulatorze ani na realnym telefonie. Pokazuje mi błędne dane, a właściwie ich brak.

W testowanym urządzeniach, które mają różne rozdzielczości nie było problemów z przeskalowaniem obiektów do prawidłowych pozycji. Jedyny problem, jaki napotkałem to w momencie obrócenia telefonu, gdy rozdzielczość diametralnie się zmienia, obiekty kolców nie są prawidłowo wyświetlane, przez co gra jest właściwie niemożliwa. Próbowałem to naprawić, stosując wymuszone skalowanie w kodzie dla obiektów, jednakże one dalej są źle wyświetlane.

Przedstawienie gry na filmie:

https://drive.google.com/file/d/1CD4QG1yRjwtdB3pNGnJWKyujqXQhp8I9/view?usp=share_link