WYŻSZA SZKOŁA EKONOMII I INFORMATYKI

KATEDRA INFORMATYKI

Kick That Boat

Dokumentacja techniczna

Imię i nazwisko: Mariusz GAJEWSKI

9 lutego 2024

Spis treści

Spis rysunków

1 Wstęp

1.1 Wykonawca projektu

Projekt w całości realizowany przez Mariusza Gajewskiego. Czynności można podzielić na dwie kategorie: część związna z tworzeniem gry oraz stworzenie fizycznej platformy służącej jako kontroler do ubsługi aplikacji. Do zadań związanych z tworzeniem gry należało: projektowanie, wykonanie grafik, modelowanie i teksturowanie 3D, programowanie oraz zarządzanie systemem kontroli wersji. Tworzenie platformy składało się z jej montażu oraz zaprojektowaniu sposobu na komunikację maty z slinikiem. Zostało to wykonane za pomocą mikrokontrolera Arduino UNO R3. Za nazwę gry odpowiada Milena Suchocka.

1.2 Wykorzystywane programy

1. Silnik gry: Unity 2021.3.12

   Projekt realizowany jest przy pomocy silnika gier Unity w wersji 2021.3.12f1. Zaletami tego rozwiązania w porównaniu do innych silników jest niski próg wejścia w naukę tworzenia gier, szeroka gama materiałów i narzędzi ułatwiająca pracę, stosunkowo krótki czas kompilacji oraz duże możliwości graficzne bez wygórowanych wymagań systemowych

   Wersja silnika obarczona numerem 2021.3.12f1 zawiera wiele poprawek i nowości w porównaniu do poprzednich wersji, a także jest oznaczona jako Long-Term Support (LTS), co w oprogramowaniu świadczy o długim czasie wspierania danej wersji przez deweloperów. Lecz jak w każdej wersji pojawiają się różnego rodzaju błędy, a do bardziej znaczących można zaliczyć brak możliwości cofnięcia zmiany kolejności obiektów w oknie hierarchii, a także dłuższy czas kompilacji shaderów w porównaniu do poprzednich wersji.

2. Modelowanie i Tekstury: Blender 3.6

   Do tworzenia modeli 3D oraz do ich teksturowania został użyty program blender w wersji 3.6. Blender to darmowe oprogrmaowanie na licencji open-source używane przez amatorów grafiki jak i przez największe korporacje działające w tej dziedzinie. Spowodowane jest to dostępnością programu i jego możliwościami sięgającymi od prostego modelowania do tworzenia pełnoprawnych animacji komputerowych.

   Główną nowością w wersji 3.6 blendera, również oznaczonej LTS, są tzw. "węzły symulacji". Umożliwiają one tworzanie zaawansowanch symulacji w czasie rzeczywistym za pomocą węzłów, tym samym ujednolicając tok pracy do tworzenia materiałów i geometrii. Wśród nowości i poprawek w kwestji projektu, ważnym usprawnieniem w nowej wersji

progrmau jest szybszy eksport modelów do formatu FBX oraz zoptymalizowany algorytm pakowania map UV.

3. Programowanie: JetBrains Rider 2023.1.1

   Część programistyczna projektu została wykonana przy pomocy IDE Rider w wersji 2023.1.1. Jest to obszerne środowisko programistyczne umożliwiające tworzenie aplikacji opierających się na platformie .NET, w typadku Unity jest to język C#. Zaletą tworzenia skryptów w Riderze jest rozwinięta integracja z silnikiem, przez co programiście udostępnionych jest wiele funkcji i narzędzi ułatiwających pracę, takie jak możliwość włączania i pauzowania gry z poziomu IDE, podgląd jakie elementy w grze używają poszczególnych klas i pól, analizę wydajności poszczególnych lini kodu, podpowiedzi do specyficznych dla Unity funkcji i wiele innych.

4. Programowanie Arduino: Arduino IDE 2.2.1

   Do komunikacji z platformami do sterowania, a grą, został użyty mikrokontroler Arduino UNO. Aby móc go zaprogramować, firma Arduino udostępnia środowisko programistyczne do tego celu. Umożliwia ono programowanie płytki w wariancie języka C++, jednyne co jest wymagane od użytkownika to znajomość rodzaju płytki oraz podłączenie jej do odpowieniego portu USB komputera.

5. Taskboard: Obsidian

   Do zarządzania projektem, tworzeniem notatek i kontrolowania zadań, została użyta aplikacja Obsidian. Jest to aplikaja do tworzenia notek pozwalająca w wygodny sposób je organizować, łączyć, tworzyć diagramy i wiele więcej. Notatki zapisywane są w formacie Markdown, co pozwala na łatwą edycję tekstu oraz otwieranie plików z poza aplikaji. Do Obsidianu można dodawać wiele wtyczek udostępnionych przez producenta jak i tworzonych przez społeczność użytkowników. W zarządzaniu projektem kluczową rolę miała wtyczka Kanban, pozwalająca tworzyć tablice i przemieszczać zadania pomiędzy nimi.

6. Kontrola wersji: GitHub i Sourcetree

   Ważnym elementem tworzenia projektów informatycznych i zarządzania nimi jest kontrola wersji. W celu został użyty GitHub. Jest to najpopularniejsze rozwiązania, pozwalające przechowywać projekt w zdalnym rerpozytorium, tworzyć commity, cofać je w razie błędów, pracować na wielu branchach, tagować commity, wgrywać różne wersje buildów i wiele więcej. GitHub jest bardzo dobrym rozwiązaniem do solowych projektów, lecz jego główną siłą jest opcja współpracy przez wielu członków zespołu nad jednym projektem. Z kontroli wersji można korzystać w pełni za pomocą konsoli systemu, lecz znacznie wygodniejszym sposobem jest użycie graficznego środowiska. Istnieje wiele z nich, w tym projekcie został użyty program

Sourcetree firmy Altasian. Oferuje on prosty interfejs użytkownika, jednocześnie posiadając szereg funkcji dla zaawansowanych użytkowników GitHuba.

7. Grafika 2D: Krita

   Do stworzenia elementów interfejsu użytkownika został użyty program Krita. Jest to program do grafiki rastrowej, chętnie używany przez cyfrowych artystów do tworzenia rysunków i grafik. Posiada czytelny interfejs użytkownika, niski próg wejścia i wiele narzędzi rysowniczych z możliwością ich dostosowania lub dodania własnych. Ze względu na skalowalność grafik, standardową drogą w tworzeniu elementów UI jest grafika wektorowa, lecz w tym wypadku gra odtwarzania będzie tylko na jednym ekranie, więc różnice w rozmiarze ikon na różnych rozdzielczościach nie będą problemem.

1.3 Użyte narzędzia

1. Cinemachine: Paczka zawierająca szeroki zakres rozbudownych kamer z dużymi możliwościami dostosowywania ich parametrów i ruchów;
2. Input System: System do ubsługi klawiatury i myszki;
3. JetBrains Rider Editor: Paczka umożliwiająca komunikację silnika ze środowiskiem JetBrains Rider;
4. Post Processing: Zestaw narzędzi do post-processów, łatwo dostosowywalnych efektów wizualnych;
5. Shader Graph: Narzędzie do tworzenia shaderów za pomocą węzłów;
6. Terrain Tools: Paczka rozwijająca możliwości dostosowania terenów na scenach w grze;
7. TextMeshPro: Zestaw elementów rozwijających podstawowe funcje interfejsu użytkownika;
8. Unity UI: Narzędzia do tworzenia interfejsu użytkownika;
9. Universal RP: Jedna z trzech głównych metod renderowania udostępniona przez Unity. Przeznaczona do gier na umiarkowanym poziomie graficznym.

2 Dodatkowe zależności

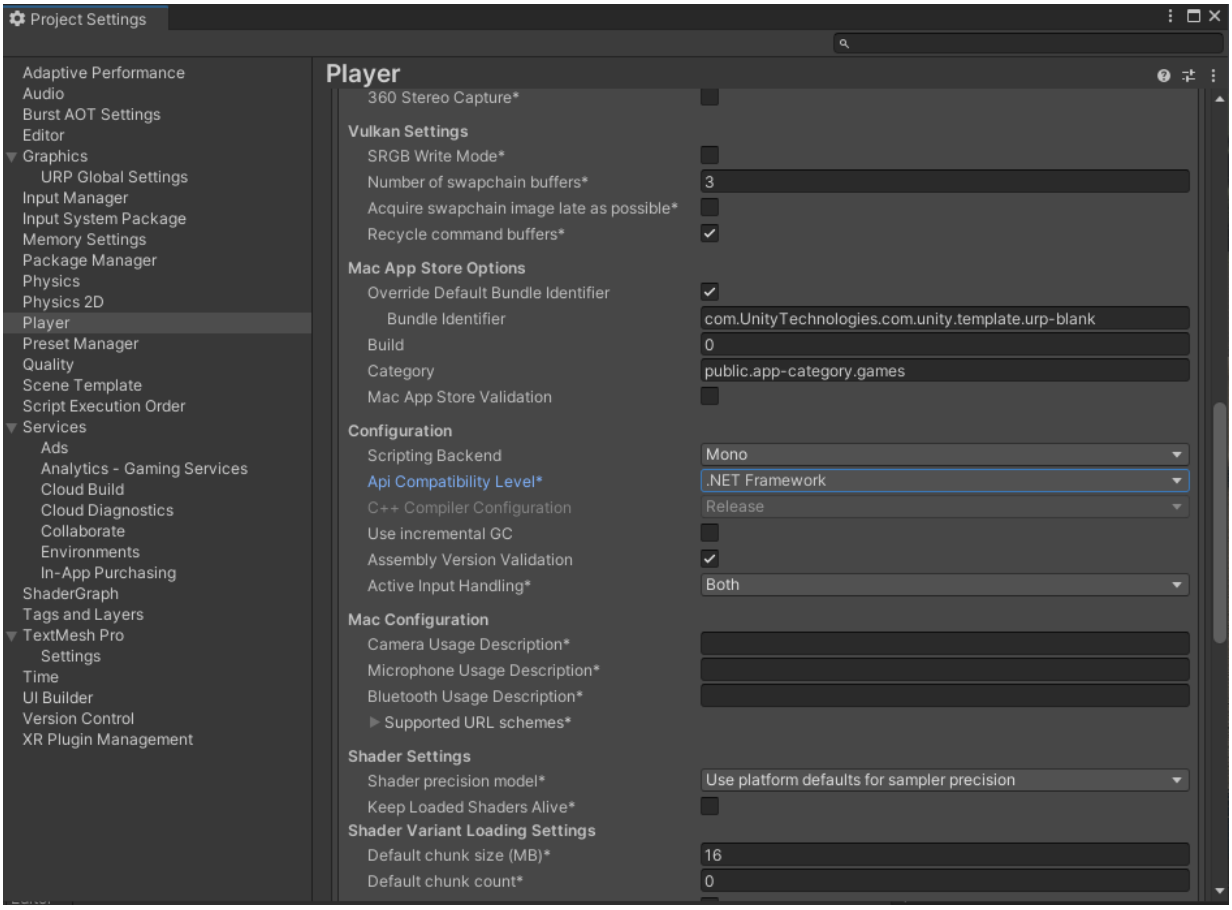
2.1 Ardity

2.1.1 Opis

Ardity to paczka do Unity pozwalająca na dwukierunkową komunikację z mikrokontrolerem Arduino poprzez port szeregowy.

2.1.2 Instalacja

W pierwszej kolejności należy zmienić poziom kompatybilności API na ".NET Framework", ponieważ domyślne środowisko nie zawiera odpowiednich klas do seryjnej komunikacji. Aby to zrobić należy wybrać "Edit->Project Settings -> Player" i w sekcji "Other settings" w częśći "Configuration" zmienić "Api Compatibility Level". 1



Rysunek 1: Zmiana poziomu kompatybilności API

W silniku, po zaimportowaniu paczki, aby dostawać wiadomości z portu szeregowego, należy dodać do sceny Prefab "SerialController" znajdujący się w folderze "Ardity -> Prefabs". W jego parametrach należy wybrać port szeregowy na który wysyłamy wiadmości oraz GameObject któty odbiera wiadomości. W owym GameObject-cie należy dodać dwie funkcje: "OnMassageArrived(string msg)", która jest wywoływana gdy zostanie otrzymana wiadmość; "OnConnectionEvent(bool success)", wykonywana podczas połączenia lub rozłączenia z portem szeregowym.

```
public class SampleMessageListener : MonoBehaviour
{
    // Invoked when a line of data is received from the serial device.
    void OnMessageArrived(string msg)
    {
        ...
    }

    // Invoked when a connect/disconnect event occurs. The parameter 'success'
    // will be 'true' upon connection, and 'false' upon disconnection or
    // failure to connect.
    void OnConnectionEvent(bool success)
    {
        ...
    }
}
```

Aby wysłać wiadomość, podczas programowania mikrokontrolera, wymaganą wartość wystarczy wypisać na port szeregowy za pomocą funkcji "Serial.println()".

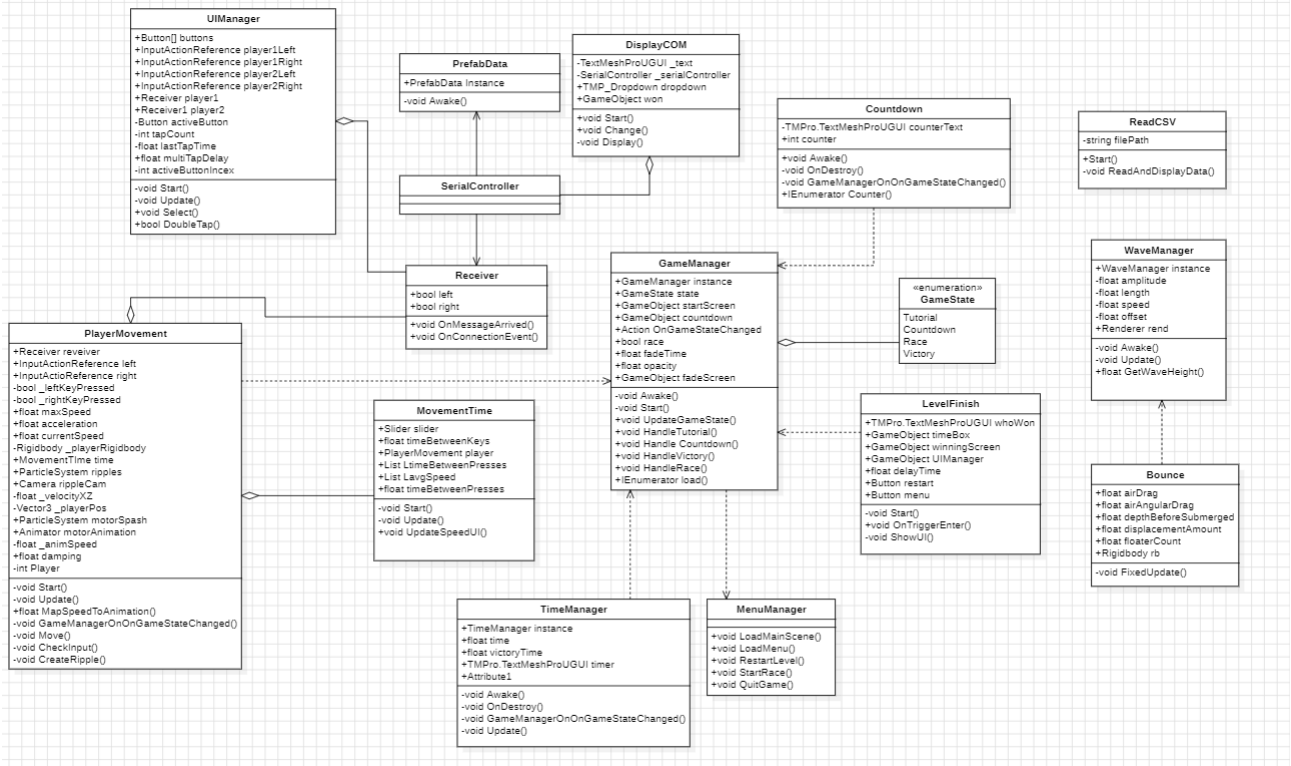
```
void setup()
{
    Serial.begin(9600);
}

void loop()
{
    Serial.println("Some message");
}
```

2.1.3 Link do oficjalnej strony:

`https://ardity.dwilches.com/`

3 Architektura

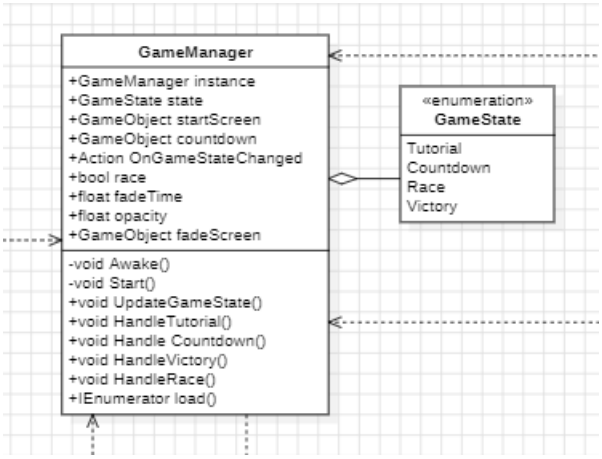
Całość części programistycznej można podzielić na *4* główne systemy oraz skrypt zarządzający stanem gry.



Rysunek 2: Diagram UML programu

3.1 Skrypt GameManager

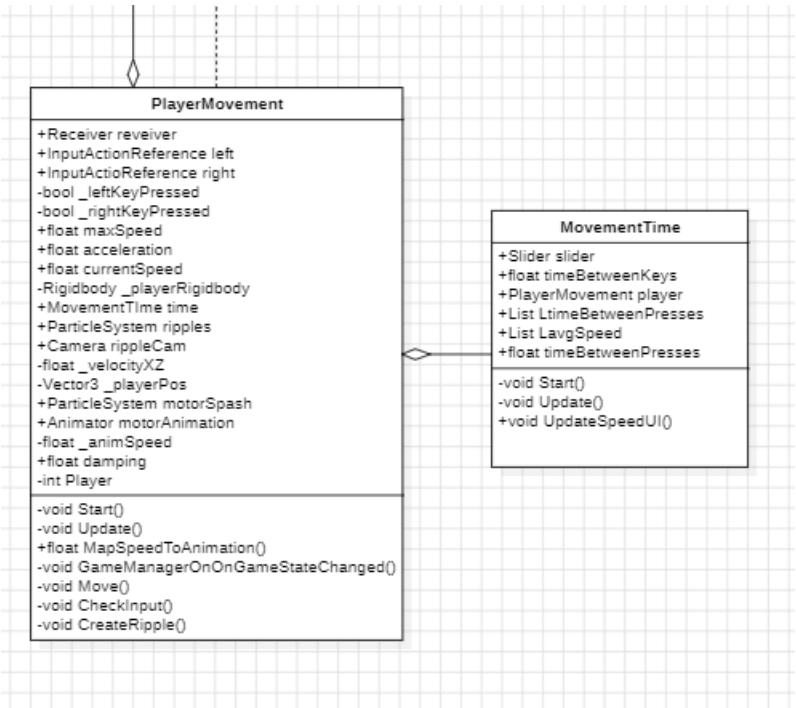
Skrypt odpowiedzialny za zmianę i zarządzanie stanami gry. W zależności od etapu rozgrywki wywoływane są inne fukcje, aktywowane są obiekty. Wyróżniamy 4 stany w kolejności ich działania: Tutorial, Countdown, Race, Victory.



Rysunek 3: GameManager

3.2 System kontolujący gracza

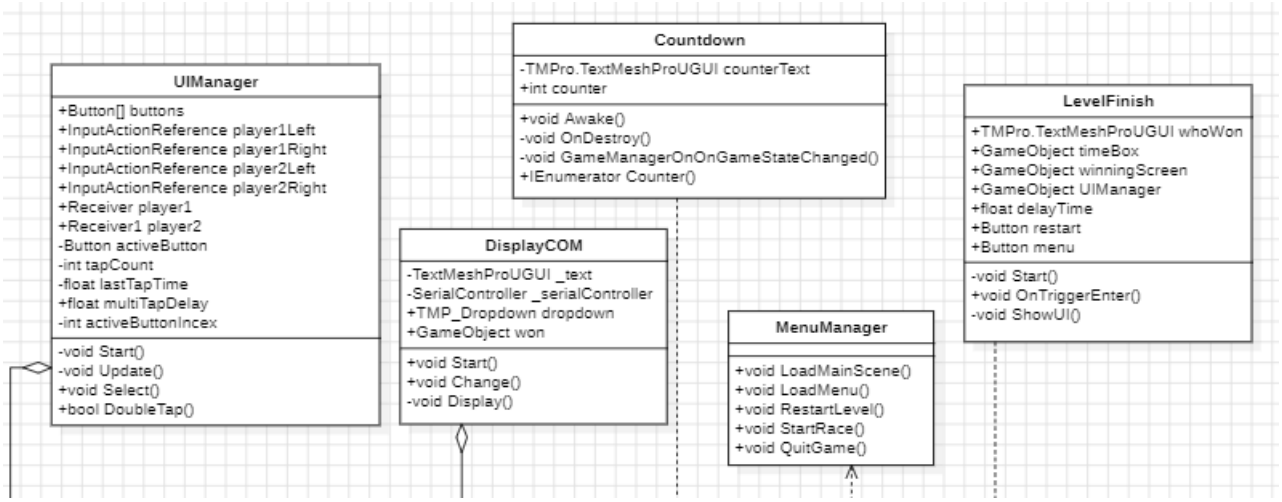
Aktywne w trakcie stanu Race, składa się z dwóch głównych skrytów: *PlayerMovement 4.1.1* i *MovementTime* 4.1.2. Pierwszy z nich pobiera dane z klasy *Receiver* 4.4.1



Rysunek 4: System kontolujący gracza

3.3 System scen i UI

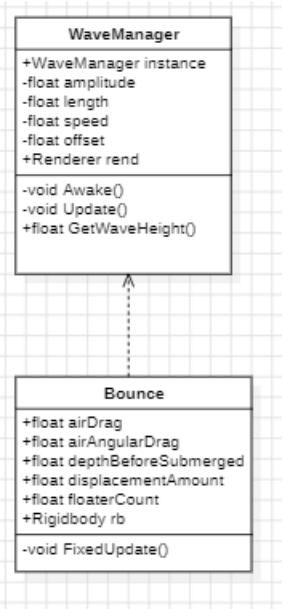
Składa się 5 klas: *UIManager* 4.2.4, *DisplayCOM* 4.2.5, *Countdown* 4.2.1, *MenuManager* 4.2.3, *LevelFinish* 4.2.2 . Aktywne są one kiedy gracz ma możliwość sterowania interfejsu użytkownika lub gdy potrzeba jest wyświetlić elementy na ekranie.



Rysunek 5: System scen i UI

3.4 System wody

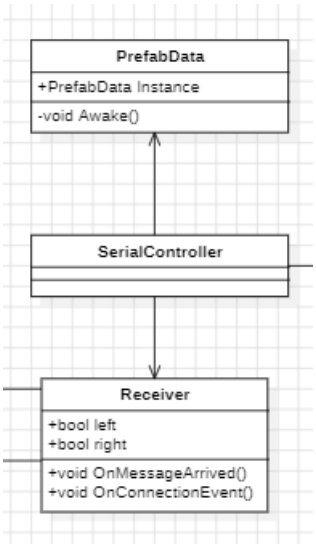
Zawiera 2 skrypty: *Bounce* 4.3.1 oraz *WaveManager* 4.3.2. Komunikuje się z klasą PlayerMovement w celu dodania mechaniki wyporności na wodzie.



Rysunek 6: System wody

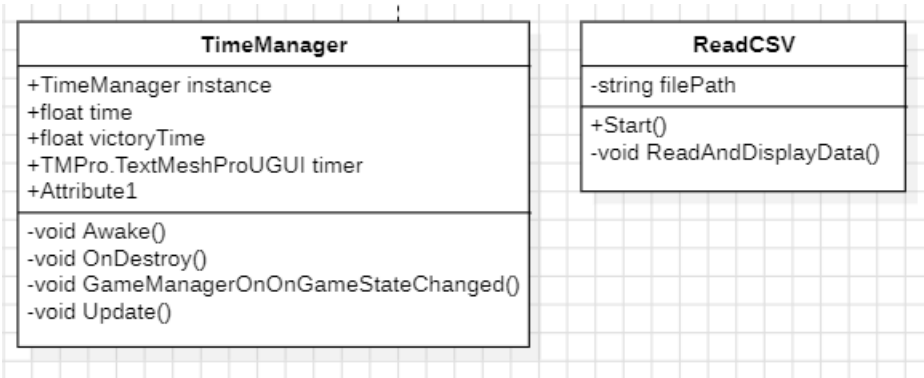
3.5 System komunikacji z Arduino

Opiera się na klasie *SerialController* będącej głównym elementem paczki Ardity 2. Posiada 2 klasy pomocnicze: *Receiver* 4.4.1 oraz *PrefabData* 4.4.2.



Rysunek 7: System komunikacji z Arduino

Do architektury należy zaliczyć jeszcze 2 skrypty nie należące do żadnego określonego systemu *TimeManager* 4.5.2 oraz *ReadCSV* 4.5.1.



Rysunek 8: Inne skrypty

4 Algorytmy

4.1 System kontrolujący gracza

4.1.1 Skrypt PlayerMovement

Skrypt jako komponent obiektu łódki mający na celu implementację systemu nadawania jej prędkości w zależności od czasu pomiędzy kliknięciami klawiszy przez gracza, pobieranymi z klasy MovementTime.

```
using UnityEngine;
using UnityEngine.InputSystem;

public class PlayerMovement : MonoBehaviour {
//Serial
public Receiver receiver;
//Input
public InputActionReference left;
public InputActionReference right;
private bool _leftKeyPressed;
private bool _rightKeyPressed;
//Movement
public float maxSpeed = 20.0f;
public float acceleration = 10.0f;
public float currentSpeed;
private Rigidbody _playerRigidbody;
public float damping = 0.2f;
//Time
public MovementTime time;
public float timeToReset = 1f;
//VFX
public ParticleSystem ripples;
public Camera rippleCam;
private float _velocityXZ;
private Vector3 _playerPos;
public ParticleSystem motorSplash;
private static readonly int Player = Shader.PropertyToID("_Player1");
public Animator motorAnimation;
private float _animSpeed;
```

```
30
31 private void Awake() {
32 /* Ustaw wartości _leftKeyPressed i _rightKeyPressed na false
33 Ustaw wartość _playerRigidbody na komponent Rigidbody
34 Włącz akcje wejściowe left i right
35 Ustaw wartość obiektu receiver na komponent Receiver instacji obiektu
       PrefabData
36 Subskrybuj event OnGameStateChanged */
37 }
38
39 private void Start() {
40 /* Stwórz okrągłą fale wokół łódki */
41 }
42
43 private void Update() {
44 /* Jeżeli stan gry to race lub tutorial
45     Wywyołaj funkcję sprawdzania inputu dla klawiatury i dla platformy
46 Zapisz pozycję gracza
47 Ustaw pozycję kamery do wyświetlania fal
48 Ustaw wartość Player w shaderze na pozycję gracza
49 Oblicz prędkość gracza
50 Jeżeli prędkość jest większa niż zero
51     Włącz efekty cząsteczkowe
52 Jeżeli jest mniejsza lub równa zero
53     Zatrzymaj efekty cząsteczkowe
54 Wywołaj funkcję MapSpeedToAnimationSpeed
55 Ustaw prędkość animacji silnika na otrzymaną wartość */
56 }
57
58 float MapSpeedToAnimationSpeed(float speed) {
59 /* Ogracznicz wartość speed od 0 do maxSpeed
60 Ustaw nową prędkość jako interoplację od 0 do 1, zależnej od
       ograniczonej prędkości i prędkości maksymalnej
61 Zwróć nową wartość */
62 }
63
64 private void Move() {
65 /* Dostosuj przyspieszenie
66 Oblicz prędkość gracza
67 Ogranicz do wartości maxSpeed
```

```
68 Oblicz siłę na podstawie prędkości
69 Dodaj siłę do rigidbody w trybie Impulse
70 Wywołaj time.UpdateSpeedUI() */
71 }
72
73 private void CheckInput(bool left, bool right) {
74 /* Jeżeli left:
75     Ustaw _leftKeyPressed na true
76     Jeżeli _rightKeyPressed
77         Wywołaj Move()
78     Ustaw _rightKeyPressed na false
79     Wyzeruj zmienną time.timeBetweenKeys
80 Jeżeli right i _leftKeyPressed:
81     Wywołaj Move()
82     Ustaw _rightKeyPressed na true
83     Ustaw _leftKeyPressed na false
84     Wyzeruj zmienną time.timeBetweenKeys
85 Jeżeli time.timeBetweenKeys jest większe od timeToReset:
86     Ustaw _leftKeyPressed i _rightKeyPressed na false
87     Wyzeruj time.timeBetweenKeys i currentSpeed
88 */
89 }
90
91 private void CreateRipple(int start, int end, int delta, float speed,
     float size, float lifetime) {
92 /* Emituj system cząsteczkowy fal na wodzie w zależności od podanych
     parametrów i pozycji gracza */
93 }
94 }
```

4.1.2 Skrypt MovementTime

Skrypt liczący czas pomiędzy kliknięciami oraz kontolujący suwak prędkości.

```
1 using System;
2 using System.Collections;
3 using System.Collections.Generic;
4 using System.Linq;
5 using UnityEngine;
```

```
using UnityEngine.UI;

public class MovementTime : MonoBehaviour {
    public Slider slider;
    public float timeBetweenKeys = 0;
    public PlayerMovement player;
    public List<float> LtimeBetweenPresses = new List<float>();
    public List<float> LavgSpeed = new List<float>();
    public float timeBetweenPresses;
    public float avgSpeed;

    private void Start() {
  /* Ustaw maksymalną wartośc suwaka prędkości na maksymalną prędkość
     gracza (player.maxSpeed) */
    }

    private void Update() {
      /* Dodaj do timeBetweenKeys czas od ostatniej klatki - wartość
         Time.deltaTime
      Jeżeli player.currentSpeed <= 0
        Odejmi 1 od slider
    Dodaj wartości timeBetweenKeys i player.currentSpeed do odpowiednich
        list w celu zbierania danych analitycznych
    Ustaw wartości timeBetweenPresses i avgSpeed jako śrenią
       odpowiednich list */
    }

    public void UpdateSpeedUI() {
    /* Dodaj do wartości slider obecną prędkość gracza podzieloną przez
       prędkość maksymalną */
    }
}
```

4.2 System scen/UI

4.2.1 Countdown

Skrypt odpowiedzialny za odliczanie czasu przed rozpoczęciem gry.

```
using System.Collections;
using UnityEngine;

public class Countdown : MonoBehaviour {
private TMPro.TextMeshProUGUI counterText;
[SerializeField] public int counter = 3;

void Awake() {
/* Zasubskrybuj event do zmiany stanu gry
Przypisz komponent TextMeshPro do wyświetlania odliczania */
}

private void OnDestroy() => GameManager.OnGameStateChanged -=
    GameManagerOnOnGameStateChanged;

private void GameManagerOnOnGameStateChanged(GameState state) {
/* Jeżeli stang gry to "Countdown", rozpoczni korutyne "Counter()"
Jeżeli stan gry to "Race", wyłącz komponent do wyświetlania odliczania *
    /
}

IEnumerator Counter() {
/* Dopóki zmienna "counter" jest większa lub równa 1:
    Wypisz ją na ektan
    Odczekaj 1 sekundę
    Odejmij 1
Zmień stan gry na "Race" */
}
}
```

4.2.2 LevelFinish

Skrypt odpowiedzialny za obsługę zakończenia gry, wywoływany, gdy któryś z graczy dotrze na metę.

```
using UnityEngine;
using UnityEngine.UI;
using System.IO;
using System;
public class LevelFinish : MonoBehaviour
{
public TMPro.TextMeshProUGUI whoWon;  //Text z informacją któty gracz
    wygrał
public GameObject timeBox;  //Okno wyświetlające czas
public GameObject winningScreen;  //Elemnt UI z informacją który gracz
    wygrał
public GameObject UIManager;  //Skrypt obsługujący sterowanie UI
public float delayTime = 1.5f;
public Button restart;
public Button menu;
[SerializeField] private string fileName = "data.csv";

private void Start()   //Wyłącz obiekty restart i menu

void OnTriggerEnter(Collider other)  {
/*Ustaw kolor tekstu w zależności od gracza
Zmień stan gry na "Victory"
Włącz objekt "winningScreen"
Włącz text "whoWon"
Wyłącz object "timeBox"
Włącz objekt UIManager
Jeżeli wygrał jeden z graczy
    Wypisz który wygrał
Wpisz do pliku czas wygranej używając funkcji WriteToCSV*/
}
private void ShowUI() // Włącz obiekty UIManager, restart i menu
private void WriteToCSV(string data)  {
/* Jeżeli plik istnieje:
        Używając StreamWriter dopisz do pliku otrzymane dane
    Lub:
```

```
34          Wywołaj CreateFile() */
35 }
36
37 private void CreateFile(string filePath)  //Używając StreamWriter stwórz
       plik
```

4.2.3 MenuManager

Skrypt odpowiedzialny za zmianę scen i zamykanie aplikacji.

```
1  using UnityEngine;
2  using UnityEngine.SceneManagement;
3  public class MenuManager : MonoBehaviour {
4  //Załaduj główną scenę
5  public void LoadMainScene()
6  //Załaduj scenę menu
7  public void LoadMenu()
8  //Załaduj ponownie scenę
9  public void RestartLevel()
10 //Zmień stan gry na Race
11 public void StartRace()
12 //Zamknij aplikację
13 public void QuitGame()
14 }
```

4.2.4 UIManager

Skrypt odpowiedzialny za sterowanie interfejsem użytkownika. Proces sterowania dla gracza następuje w pełni za pomocą platformy sterującej, jedynie do wyboru portu szeregowego należy użyć myszki. Naciśnięcie lewego klawisza powoduje przesunięcie aktywnego przycisku w lewo, analogicznie, prawy przycisk przesuwa fokus interfejsu w prawo. Podwójne naciśnięcie po odpowieniej stronie aktywuje przycisk.

```
1  using System;
2  using UnityEngine;
3  using UnityEngine.InputSystem;
4  using UnityEngine.UI;
```

```
using UnityEngine.EventSystems;

public class UIManager : MonoBehaviour
{
public Button[] buttons;

public InputActionReference player1Left;
public InputActionReference player1Right;
public InputActionReference player2Left;
public InputActionReference player2Right;

public Receiver player1;
public Receiver2 player2;
private Button activeButton;
private int tapCount = 0;
private float lastTapTime = 0f;
public float multiTapDelay = 0.8f;

private int activeButtonIndex;

private void Start()  {
/*Wybierz pierwszy przycisk
Ustaw go jako aktywny
Włącz akcje inputu
Podłącz klase Receiver dla obu graczy*/
}

private void Update()
{
/*Pobierz index aktywnego przycisku
Jeżeli lewy przycisk pierwszego gracza został aktywowany (klawiaturą lub
    matą):
    Jeżeli aktywny przycisk ma index 0:
        Nadal ten sam przycisk
    Lub:
        Ustaw aktywny przycisk na przycisk o inedexie mniejszym o 1
        Wybierz ten przycisk
Jeżeli prawy przycisk pierwszego gracza został aktywowany (klawiaturą
   lub matą):
    Jeżeli aktywny przycisk ma ostatni index :
```

```
43          Nadal ten sam przycisk
44      Lub:
45          Ustaw aktywny przycisk na przycisk o inedexie większym o 1
46          Wybierz ten przycisk
47  Jeżeli lewy przycisk drugiego gracza został aktywowany (klawiaturą lub
        matą):
48      Jeżeli aktywny przycisk ma index 0:
49          Nadal ten sam przycisk
50      Lub:
51          Ustaw aktywny przycisk na przycisk o inedexie mniejszym o 1
52          Wybierz ten przycisk
53  Jeżeli prawy przycisk drugiego gracza został aktywowany (klawiaturą lub
        matą):
54      Jeżeli aktywny przycisk ma ostatni index:
55          Nadal ten sam przycisk
56      Lub:
57          Ustaw aktywny przycisk na przycisk o inedexie większym o 1
58          Wybierz ten przycisk
59
60  Jeżeli aktywny przycisk ma index 0:
61      Jeżli funkcja DoubleTap dla parametrów lewgo przycisku dla obu
            graczy zwwróci true:
62          Wywołaj Select()
63  Jeżeli aktywny przycisk ma ostatni index:
64      Jeżli funkcja DoubleTap dla parametrów prawego przycisku dla obu
            graczy zwwróci true:
65          Wywołaj Select()*/
66  }
67
68  //Aktywuj przycisk
69  void Select() => activeButton.onClick.Invoke();
70
71  bool DoubleTap(bool p1, bool p2) {
72  /* Jeżeli p1 lub p2:
73      Ustaw wartość timeSinceLastTap jako różnice obecnego czasu, a zmienn
            ą lastTapTime
74      Jeżeli ta wartość jest mniejsza lub równa zmiennej multiTapDelay:
75          Zwiększ tapCount o 1
76          Ustaw p1 i p2 na false
77      Lub:
```

```
78          tapCount = 1
79      Zmienna lastTapTIme równa obecnemu czasowi
80      Jeżeli tapCount >= 2:
81          Zwróć true
82  Zwróć false*/
83  }
```

4.2.5 DisplayCOM

Skrypt wyświetlający w menu głównym wybrany port szeregowy oraz pozwalający wybrać inny. Wartość ta przechowywana jest w pliku systemowym.

```
1  using TMPro;
2  using UnityEngine;
3  using System.IO;
4  public class DisplayCOM : MonoBehaviour  {
5  private TextMeshProUGUI _text;
6  private SerialController _serialController;
7  public TMP_Dropdown dropdown;
8  public GameObject com;
9  [SerializeField] private string fileName = "port.txt";
10 string filePath;
11 void Start()  {
12 /*Utwórz systemową ścierzkę do pliku
13 Przypisz do _text komponent TextMeshProUGUI
14 Przypisz do _serialController komponent SerialController obiektu com
15 Wywołaj funkcję ReadFile()
16 Wywołaj funkcję Display()*/
17 }
18 public void Change()  {
19 /*Zmienna port równa aktywnej wartości menu Dropdown
20 Parametr _serialController.portName równy tekstowej warości portu
21 Przypisz nazwę portu do paramentru yourValueChanged instancji PrefabData
22 Ustaw _serialController.portName na wartość yourValueChanged
23 Wywołaj funkcję WriteToFile
24 Wywołaj funkcję Display()*/
25 }
26 private void Display()
27 //Wypisz tekst z aktualnie wybranym portem
```

```
28 }
29 private void WriteToFile(int port)  {
30 /*Jeżeli plik istnieje:
31     Używając StreamWriter nadpisz go z zumerem portu
32 Lub:
33     Wywołaj CreateFile()*/
34 }
35 private void ReadFile()  {
36 /*Jeżeli plik istnieje:
37     Przypisz _serialController.portName odczytaną linię z pliku
38 Lub:
39     Wywołaj CreateFile() */
40 }
41 private void CreateFile(string filePath)  {
42 //Używając StreamWriter stwórz plik
43 }
44 }
```

4.3 System wody

4.3.1 Bounce

Skrypt Bounce używany jest na obiektach unoszących się na powierzchni wody za pomocą fizyki. W tym celu dodaje do nich siłę wyporności.

```
1 using System;
2 using System.Collections;
3 using System.Collections.Generic;
4 using UnityEngine;
5
6 public class Bounce : MonoBehaviour
7 {
8 public float airDrag = 0f;
9 public float airAngularDrag = 0.05f;
10 public float depthBeforeSubmerged = 1f;
11 public float displacementAmount = 3f;
12 public float floaterCount = 1;
13
```

```
14 public Rigidbody rb;
15
16 private void FixedUpdate()
17 {
18 /*
19 Dodaj siłę grawitacji do łódki
20 Pobierz informację o wysokości fali ze skryptu WaveManager
21 Jeżeli "floater" jest pod poziomem wody
22     Dodaj siły wyporności
23 */
24 }
25 }
```

4.3.2 WaveManager

Skrypt pobiera z materiału informację o falach na wodzie i na ich podstawie oblicza wysokość fal.

```
1 using System;
2 using System.Collections;
3 using System.Collections.Generic;
4 using UnityEngine;
5 public class WaveManager : MonoBehaviour
6 {
7 public static WaveManager instance;
8
9 private float amplitude;
10 private float length;
11 private float speed;
12 private float offset;
13
14 public Renderer rend;
15
16 private void Awake()
17 {
18 /*
19 Jeżeli ten obiekt nie istnieje
20 Przypisz ten obiekt jako instance
21 Lub
```

```
Zniszcz ten obiekt
Przypisz zmiennej rend komonent Renderer
*/
}
private void Update()
{
/*
Dodaj do offset Time.deltaTime * speed
Pobierz z właściwości materiału odpowednie wartości dla amplitude,
    length, speed
*/
}
public float GetWaveHeight(float _x)
{
/*
Wysokość fali wynosi amplitude * Mathf.Sin(_x / length + offset)
*/
}
}
```

4.4 System komunikacji z Arduino

4.4.1 Receiver

Klasa Receiver służy jako pośrednik między paczką Ardity, a kontrolerm gracza. Znajduje się w niej funkcja wymagana do odbierania odczytywanych wiadomości z portu szeregowego.

```
using UnityEngine;

public class Receiver : MonoBehaviour  {
public bool left;
public bool right;

void OnMessageArrived(string msg)  {
    /*Ustaw left i right na false
    Jeżli msg wynosi "1":
        Ustaw left na true
    Jeżeli msg wynosi "2":
```

```
12          Ustaw right na true */
13  }
14  }
```

4.4.2 PrefabData

Prosta klasa implementująca na obiekcie ze skryptami SerialControler oraz Receiver wzorzec Singleton, oraz powodująca że zostaje on niezmieniony podczas zmiany scen.

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class PrefabData : MonoBehaviour  {
6  public static PrefabData Instance;
7  public string yourValueChanged;
8
9  private void Awake()  {
10     /* Jeżeli ten obiekt nie istnieje
11          Przypisz ten obiekt jako instance
12        Lub
13          Zniszcz ten obiekt */
14 }
```

4.5 Pozostałe skrypty

4.5.1 ReadCSV

Skrypt obsługujący tablicę najlepszych wyników. Odczytuje on zawartość pliku data.csv, sortuje oraz wyświetla 10 pierwszych wyników.

```
1  using UnityEngine;
2  using System.IO;
3  using System;
4  using System.Collections.Generic;
5  using System.Linq;
6  using TMPro;
```

```
7
8  public class ReadCSV : MonoBehaviour  {
9  [SerializeField] private string fileName = "data.csv";
10 public TextMeshProUGUI text;
11
12 void Start()  {
13     /*Przypisz zmiennej text komponent TextMeshProUGUI
14        Wywołaj funkcję ReadAndDisplayData()*/
15 }
16
17 private void ReadAndDisplayData()  {
18     /* Utwórz zmienną filePath łączącą domyślną ścieżkę systemową
          parametru właściwości klasy Application, z ścieżką zawartą w
          zmiennej fileName
19     Jeżeli plik istnieje
20         Odczytaj wszystkie linie
21         Konweruj zawartość na float i posortuj rosnąco
22         Wyświetl pierwsze 10 lini jako zawartość zmiennej text
23     Lub
24         Wywołaj CreateFile() */
25 }
26 private void CreateFile(string filePath)  {
27     /* Używając StreamWriter, stwórz plik */
28 }
29 }
```

4.5.2 TimeManager

Skrypt na bazie wzorca Sigleton, odpowiedzialny za liczenie i wyświetlanie czasu podczas rozgrywki.

```
1 using System;
2 using UnityEngine;
3
4 public class TimeManager : MonoBehaviour
5 {
6     public static TimeManager instance;
7     public float time;
8     public float victoryTime;
```

```
    public TMPro.TextMeshProUGUI timer;
    public TMPro.TextMeshProUGUI finalTimeText;

    private void Awake()  {
      /* Jeżeli ten obiekt nie istnieje
         Przypisz ten obiekt jako instance
       Lub
         Zniszcz ten obiekt
       Zasubskrubuj event OnGameStateChanged */
    }

    private void OnDestroy() /*Odsubskrybuj event OnGameStateChanged*/

    private void GameManagerOnOnGameStateChanged(GameState state)  {
      /* Pobierz informacje o stanie gry
    Jeżeli stan to:
      Tutorial:
        Nic nie rób
      Countdown:
        Ustaw time na 0
      Race:
        Nic nie rób
      Victory:
        Przypisz zmiennej victoryTime wartość time
        Wypisz końcowy czas */
   }

    private void Update()  {
      /* Jeżeli stan gry to race:
         Dodaj do time czas od ostaniej klatki
         Wypisz time*/
    }
}
```

4.5.3 GameManager

Skrypt na bazie wzorca Sigleton, zarządza stanami gry: Tutorial, Countdown, Race i Victory.

```
using System;
```

```
using UnityEngine;

public class GameManager : MonoBehaviour  {
    public static GameManager instance;
    public GameState state;
    public GameObject startScreen;
    public GameObject countdown;
    public static event Action<GameState> OnGameStateChanged;
    public bool race;

    private void Awake() // Ten objekt jest jedyną instancja tej klasy

    private void Start()  //Ustaw stan gry na Tutorial

    public void UpdateGameState(GameState newState)  //Wywołaj
       odpowiednie funcje z wależności od stanu gry i aktualizuj stan

    void HandleCountdown()  //Wyłącz obiekt startScreen i włącz obiekt
       Countdown

    void HandleVictory()  // Ustaw wartość zmiennej race na false

    void HandleRace()  // Ustaw wartość zmiennej race na true
}

public enum GameState {
    Tutorial,
    Countdown,
    Race,
    Victory
}
```

5 Procesy

5.1 Tworzenie gry

Proces tworzenia gry składał się z następujących kroków:

- Pierwszym krokiem było przetestowania paczki Ardity. Test polegał na poruszaniu obiektu w Unity przy pomocy przycisku podłączonego do płytki Arduino;
- Kolejnym etapem było uzależnienie prędkości poruszania się gracza od odstępu czasowego pomiędzy naciśnięciami odpowiednich klawiszy na klawiaturze;
- Następnym krokiem było dodanie początku poziomu z odliczaniem do startu oraz końca, przy którym na ekranie wyświetlał się czas ukończenia wyścigu;
- Potem zostały dodane modele 3D, teren, tekstury oraz "interaktywna" woda przy pomocy shaderu. Częścią systemu wody jest mechanika wyporności obiektów - elementy unoszą się na wodzie za pomocą fizyki;
- Gdy podstawowa warstwa wizualna została ukończona, nastąpiła integracja paczki Ardity, kontrolowana przez testową wersję platformy, składającą się z Arduino i dwóch przycisków. Aby spełnić cel wyścigu, został dodany drugi gracz, wraz ze wszystkimi elementami potrzbnymi do jego obsługi;
- Do ostanich kroków należało ulepszenie wartstwy UI, w celu pełnego sterowania z poziomu maty oraz atrakcyjnego wyglądu, oraz elementów takich jak oświetlenie, post-procesy, efekty cząsteczkowe oraz animacje

5.2 Tworzenie platformy sterującej

Proces tworzenia platformy sterujacej polegał na:

- Inspiracją do stworzenia platformy sterującej były maty do tańczenie używane w salonach Arcade w grach takich jak Dance Dance Recolution.
- Zasada działania platformy jest podobna do owych mat. Budowę można podzielić na 2 segmenty. Na spodzie znajdują się 2 blaszki z przewodami biegnącymi do mikrokontrolera. Jeden z przewodów podłączony jest do pinu uziemienia, drugi do pinu cyfrowego. Ważnym krokiem jest ustawienie owego pinu w trybie INPUT_PULLUP.
- Uogólniając, działanie przycisków można porównać do klasycznych przycisków typu switch. Na górnym elemencie znajduję się jedna długa blaszka. W chwili nadepnięcia na matę górny element zwiera dolne blaszki, przez co stan pinu na mikrokontolerze zmienia się z wysokiego na niski.
- W tym momencie Arduino wysyła wiadomość na port szeregowy komputera, a skrypt odpowiedzialny za komunikację po stronie Unity, odczytuję odebraną wiadomość.

6 Potencjalne kierunki rozwoju

W celu udoskonalenia projektu można wprowadzać dalsze poprawki w warstwie wizualej. Biorąc pod uwagę cel projektu, ma to bardzo ważne znaczenie, aby gra była jak najładniejsza. Innym kierunkiem rozwoju jest dodanie innych opcji sterowania, na przykład określona kolejność klawiszy w celu pokonania przeszkody.