Zaawansowane Techniki Programowania Gier Komputerowych

Raport - Projekt

Autor: Patryk Gregorczuk

1. Zasady gry

W grze wcielamy się w rolę kapitana statku kierując nim w trybie pierwszoosobowej kamery. Pierwszą misją jest zejście na ląd i zdobycie rumu dla zbuntowanej załogi. Po okiełznaniu rozzuchwalonych marynarzy gracz może sterować statkiem, a jego kolejnym celem jest przepłynięcie trasy wyznaczonej przez unoszące się na wodzie tajemnicze pudełka, które w różny sposób oddziałują na statek.

2. Specyfikacja wewnętrzna

Pierwsze trzy skrypty, które należy omówić służą do sterowania kamerą w trybie pierwszoosobowym. W pierwszym przypadku mamy do czynienia z obracaniem samej kamery natomiast w drugim obracamy również postać gracza. Podstawowym skryptem jest MouseLook, który dokonuje podstawowych przeliczeń, natomiast dziedziczące po nim skrypty PlayerMouseLook oraz ShipMouseLook implementują konkretne zachowania w zależności od aktualnego stanu gry.

MouseLook.cs

```
using UnityEngine;

public class MouseLook : MonoBehaviour
{
    public float sensitivity = 50.0f;

    protected float mouseX;
    protected float mouseY;

    protected float xRotation;
    protected float yRotation;

    protected void Start()
    {
        Cursor.lockState = CursorLockMode.Locked;

        xRotation = transform.localRotation.eulerAngles.x;
        yRotation = transform.localRotation.eulerAngles.y;
    }

    protected void Update()
    {
        mouseX = Input.GetAxis("Mouse X") * sensitivity * Time.deltaTime;
        mouseY = Input.GetAxis("Mouse Y") * sensitivity * Time.deltaTime;

        xRotation -= mouseY;
        xRotation = Mathf.Clamp(xRotation, -90.0f, 90.0f);

        yRotation -= mouseX;
    }
}
```

PlayerMouseLook.cs

```
using UnityEngine;

public class PlayerMouseLook : MouseLook
{
	public Transform player;

	protected new void Update()
	{
		if (GameModeSwitch.IsHumanoidControlled)
		{
			base.Update();

			transform.localRotation = Quaternion.Euler(xRotation,
-yRotation, 0.0f);

			player.Rotate(Vector3.up * mouseX);
		}
	}
}
```

ShipMouseLook.cs

```
using UnityEngine;

public class ShipMouseLook : MouseLook
{
	protected new void Update()
	{
		if (!GameModeSwitch.IsHumanoidControlled)
		{
			base.Update();

			transform.localRotation = Quaternion.Euler(xRotation,
-yRotation, 0.0f);
		}
	}
}
```

Kolejne dwa istotne skrypty to GameModeSwitch oraz DayAndNightSwitch. Pierwszy z nich odpowiada za schodzenie na ląd oraz powrót na statek oraz za wyświetlenie guzika z podpowiedzią, który klawisz na klawiaturze nacisnąć. Przełączanie polega na ustawianiu flagi isActive dla odpowiednich kamer. Na początku gry aktywna jest kamera pokładowa. Skrypt DayAndNightSwitch odpowiada za przełączania dnia i nocy. Działanie opiera się na ustawianiu odpowiednich wartości dla shaderów wody (metoda AdjustWater), globalnego oświetlenia (metoda AdjustLight) oraz ustawienie kolorów dla shadera nieba (metoda AdjustSky). Podobnie jak w przypadku GameModeSwitch wyświetla on również guzik z informacją o klawiszu.

GameModeSwitch.cs

```
using UnityEngine;

public class GameModeSwitch : MonoBehaviour
{
	public static bool IsHumanoidControlled { get; set; }

	public Camera shipCamera;
	public Camera humanCamera;

	private void OnGUI()
	{
		if (GUI.Button(new Rect(10, 40, 200, 25), (IsHumanoidControlled ? "Go
back to the ship" : "Go ashore") + " (C)"))
		{
			ToggleGameMode();
		}
	}

	private void Update()
	{
		if (Input.GetKeyDown(KeyCode.C))
		{
			ToggleGameMode();
		}
	}

	private void ToggleGameMode()
	{
		IsHumanoidControlled = !IsHumanoidControlled;

		SwitchCamera();
	}

	private void SwitchCamera()
	{
		humanCamera.enabled = IsHumanoidControlled;
		shipCamera.enabled = !IsHumanoidControlled;
	}
}
```

DayAndNightSwitch.cs

```
using UnityEngine;

public class DayAndNightSwitch : MonoBehaviour
{
	public static bool IsDay = true;

	public GameObject water;
	public Shader waterAtDayShader;
	public Shader waterAtNightShader;

	public Light light;
	public Flare sunFlare;
```

```
	public GameObject horizen;

	private void OnGUI()
	{
		if (GUI.Button(new Rect(10, 10, 200, 25), "Toggle day/night (N)"))
		{
			ToggleDayAndNight();
		}
	}

	private void Update()
	{
		if (Input.GetKeyDown(KeyCode.N))
		{
			ToggleDayAndNight();
		}
	}

	private void ToggleDayAndNight()
	{
		IsDay = !IsDay;

		AdjustWater();

		AdjustLight();

		AdjustSky();
	}

	private void AdjustWater()
	{
		if (water)
		{
			water.GetComponent<MeshRenderer>().material.shader = IsDay ?
waterAtDayShader : waterAtNightShader;
		}
	}

	private void AdjustLight()
	{
		if (light)
		{
			if (sunFlare)
			{
				light.flare = IsDay ? sunFlare : null;
			}

			light.intensity = IsDay ? 1.0f : 0.1f;
		}
	}

	private void AdjustSky()
	{
```

```
            if (horizen)
            {
                  var horizonMaterial =
horizen.GetComponent<MeshRenderer>().material;

                  if (ColorUtility.TryParseHtmlString(IsDay ? "#0015BF00" :
"#000000C0", out Color colorLevel1))
                  {
                        horizonMaterial.SetColor("_Level1Color", colorLevel1);
                        horizonMaterial.SetFloat("_Level1", IsDay ? 10000 :
1000);
                  }

                  if (ColorUtility.TryParseHtmlString(IsDay ? "#93C7FDFF" :
"#101020FF", out Color colorLevel0))
                  {
                        horizonMaterial.SetColor("_Level0Color", colorLevel0);
                  }
            }
      }
}
```

Dwa kolejne skrypty odpowiadają za poruszanie się na lądzie oraz na morzu. Są to kolejno PlayerMovement oraz ShipPhysics. Pierwszy z nich odpowiada za poruszanie modelu postaci na lądzie w każdym kierunku za pomocą komponentu CharacterController oraz umożliwia podskakiwanie. Natomiast drugi wpływa na zachowanie statku poprzez wykorzystanie metod fizyki dodających siłę oraz moment siły z uwzględnieniem dodatkowych parametrów statku. Dodatkowo odpowiada on za wyświetlanie informacji o aktualnym stanie rozgrywki.

PlayerMovement.cs

```
using UnityEngine;

public class PlayerMovement : MonoBehaviour
{
      private CharacterController characterController;

      public float speed = 6.0f;
      public float jumpSpeed = 8.0f;
      public float gravity = 20.0f;

      private Vector3 movementDirection = Vector3.zero;

      void Start()
      {
            characterController = GetComponentInParent<CharacterController>();
      }

      void Update()
      {
            if (GameModeSwitch.IsHumanoidControlled)
            {
                  if (characterController.isGrounded)
```

```
                        {
                                movementDirection = transform.forward *
-Input.GetAxis("Vertical") + transform.right * -Input.GetAxis("Horizontal");
                                movementDirection *= speed;

                                if (Input.GetKeyDown(KeyCode.Space))
                                {
                                        movementDirection.y = jumpSpeed;
                                }
                        }

                        movementDirection.y -= gravity * Time.deltaTime;

                        characterController.Move(movementDirection * Time.deltaTime);
                }
        }
}
```

ShipPhysics.cs

```
using UnityEngine;

public class ShipPhysics : MonoBehaviour
{
        public float accelerationForward = 25.0f;
        public float accelerationBackward = 10.0f;
        public float accelerationSide = 15.0f;

        public float maxSpeed = 10.0f;
        public float maneuverability = 0.1f;

        private Rigidbody rigidbody;

        public Player player;

        private void Start()
        {
                rigidbody = gameObject.GetComponent<Rigidbody>();

                maneuverability = Mathf.Clamp(maneuverability, 0.01f, 0.15f);
        }

        private void OnGUI()
        {
                if (!GameModeSwitch.IsHumanoidControlled)
                {
                        if (!player.MissionCompleted)
                        {
                                var rect = new Rect(Screen.currentResolution.width / 2
- 150, Screen.currentResolution.height / 2 - 100, 300, 200);
                                var guiStyle = new GUIStyle();
                                guiStyle.normal.textColor = Color.red;
                                guiStyle.fontSize = 20;
                                GUI.Label(rect, "We cannot sail without rum! Before we
go anywhere, we need to go ashore to find some barrels.", guiStyle);
```

```
                }
            }
        }

        private void FixedUpdate()
        {
            if (!GameModeSwitch.IsHumanoidControlled && player.MissionCompleted)
            {
                if (rigidbody.velocity.magnitude < maxSpeed)
                {
                    if (Input.GetKey(KeyCode.W))
                    {
                        rigidbody.AddForce(transform.forward *
Time.fixedDeltaTime * accelerationForward * 10.0f);
                    }
                    else if (Input.GetKey(KeyCode.S))
                    {
                        rigidbody.AddForce(-transform.forward *
Time.fixedDeltaTime * accelerationBackward * 10.0f);
                    }
            }

            if (Mathf.Abs(rigidbody.angularVelocity.y) < maneuverability)
            {
                if (Input.GetKey(KeyCode.D))
                {
                    rigidbody.AddTorque(transform.up * Time.fixedDeltaTime
* accelerationSide * 50.0f);
                }
                else if (Input.GetKey(KeyCode.A))
                {
                    rigidbody.AddTorque(-transform.up *
Time.fixedDeltaTime * accelerationSide * 50.0f);
                }
            }

            Debug.Log("Speed = " + rigidbody.velocity.magnitude + " Velocity = "
+ rigidbody.velocity + " Angular velocity = " + rigidbody.angularVelocity);
            }
        }
}
```

Kolejnymi skryptami są Player oraz Ship. Pierwszy odpowiada za wykrywanie kolizji z beczkami na lądzie i kontroluje ich podnoszenie, wyświetlanie i przechowywanie informacji o aktualnej ilości zebranych beczek, a także informację czy misja lądowa została zakończona. Skrypt Ship wyświetla informację o aktualnym stanie misji morskiej o aktualnej ilości zebranych skrzynek oraz udostępnia metodę zwiększającą stan zebranych skrzynek, co jest wykorzystane w kolejnym skrypcie jakim jest PickupBoxTrigger. Skrypt ten odpowiada za wykrycie kolizji ze statkiem oraz zadziałanie na niego losowym bądź zdefiniowanym efektem. Działa on operacjami fizyki na komponent RigidBody statku bądź modyfikuje parametry w skrypcie ShipPhysics. Wyróżniono efekty takie jak: chwilowe przyspieszenie/spowolnienie,

zwiększenie/zmniejszenie maksymalnej prędkości, zwiększenie sterowności, zwiększenie przyspieszenia podczas płynięcia do przodu/skręcania.

Player.cs

```
using UnityEngine;

public class Player : MonoBehaviour
{
	private int collectedBarrels = 0;
	private int barrelsToCollect = 5;

	public bool MissionCompleted
	{
		get
		{
			return collectedBarrels == barrelsToCollect;
		}
	}

	private void OnGUI()
	{
		if (GameModeSwitch.IsHumanoidControlled)
		{
			GUI.Label(new Rect(10, 70, 300, 25), $"Barrels collected: 
{collectedBarrels}/{barrelsToCollect}");

			if (MissionCompleted)
			{
				GUI.Label(new Rect(10, 100, 300, 25), $"Land mission 
completed!");
			}
		}
	}

	void OnCollisionStay(Collision col)
	{
		if (!MissionCompleted)
		{
			if (col.gameObject.tag == "Barrel")
			{
				if (Input.GetKeyDown(KeyCode.E))
				{
					collectedBarrels++;
					Destroy(col.gameObject);
				}
			}
		}
	}
}
```

Ship.cs

```
using UnityEngine;
```

```
public class Ship : MonoBehaviour
{
	private int collectedBoxes = 0;
	private int boxesToCollect = 5;

	public bool MissionCompleted
	{
		get
		{
			return collectedBoxes == boxesToCollect;
		}
	}

	private void OnGUI()
	{
		if (!GameModeSwitch.IsHumanoidControlled)
		{
			GUI.Label(new Rect(10, 70, 300, 25), $"Boxes collected:
{collectedBoxes}/{boxesToCollect}");

			if (MissionCompleted)
			{
				GUI.Label(new Rect(10, 100, 300, 25), $"Sailing
mission completed! Congratulations!");
			}
		}
	}

	public void CollectBox()
	{
		collectedBoxes++;
	}
}
```

PickupBoxTrigger.cs

```
using UnityEngine;

public class PickupBoxTrigger : MonoBehaviour
{
	public int effectId;

	private void Start()
	{
		if (effectId == -1)
		{
			effectId = Random.Range(0, 6);
		}
	}

	void OnTriggerEnter(Collider col)
	{
		if (col.tag == "Ship")
		{
```

```
                    switch (effectId)
                    {
                            case 0:
                            col.attachedRigidbody.AddForce(col.transform.forward *
20.0f, ForceMode.VelocityChange);
                            break;
                            case 1:
                            col.attachedRigidbody.AddForce(-col.transform.forward
* 20.0f, ForceMode.VelocityChange);
                            break;
                            case 2:
                            col.GetComponent<ShipPhysics>().maxSpeed *= 2.0f;
                            break;
                            case 3:
                            col.GetComponent<ShipPhysics>().maxSpeed /= 1.5f;
                            break;
                            case 4:
                            col.GetComponent<ShipPhysics>().maneuverability *=
1.25f;
                            break;
                            case 5:
                            col.GetComponent<ShipPhysics>().accelerationSide *=
1.25f;
                            break;
                            case 6:
                            col.GetComponent<ShipPhysics>().accelerationForward *=
1.25f;
                            break;
                    }

                    col.GetComponent<Ship>().CollectBox();

                    Destroy(gameObject);
            }
        }
}
```

Ostatnie trzy skrypty to FlashLightControl, CubePeriodicRotation oraz LightPulse. Pierwszy z nich odpowiada za włączania i wyłączanie obiektu z komponentem Spot Light, który ma imitować działanie latarki przyczepionej do postaci. CubePeriodicRotation odpowiada za obracanie obiektów przez dwie sekundy w zadanych odstępach czasu. Względem wersji opisanej w raporcie z laboratorium z fizyki został on wzbogacony o losowy czas początkowy dzięki czemu obiekty nie obracają się w tym samym momencie. Skrypt LightPulse odpowiada za sterowanie pulsującym światłem emitowanym przez beczki na lądzie.

FlashLightControl.cs

```
using UnityEngine;

public class FlashLightControl : MonoBehaviour
{
        private Light flashLight;
```

```
        private void Start()
        {
                flashLight = GetComponent<Light>();
        }

        private void Update()
        {
                if (Input.GetKeyDown(KeyCode.F) &&
GameModeSwitch.IsHumanoidControlled)
                {
                        flashLight.enabled = !flashLight.enabled;
                }
        }
}
```

CubePeriodicRotation.cs

```
using System.Collections;
using UnityEngine;

public class CubePeriodicRotation : MonoBehaviour
{
        private float time = 0.0f;
        private float periodTime = 2.0f;

        private bool isRotating = false;

        public float rotationTime = 2.0f;
        public float rotationSpeed = 45.0f;

        private void Start()
        {
                time = Random.Range(0.0f, periodTime);
        }

        private void Update()
        {
                time += Time.deltaTime;

                if (time > periodTime && !isRotating)
                {
                        time -= periodTime;

                        StartCoroutine(RotateForSeconds());
                }
        }

        private IEnumerator RotateForSeconds()
        {
                isRotating = true;

                float remainingRotationTime = rotationTime;

                while (true)
```

```
		{
			if (remainingRotationTime > 0.0f)
			{
				transform.Rotate(Vector3.up, Time.deltaTime *
rotationSpeed);
				remainingRotationTime -= Time.deltaTime;
				yield return null;
			}
			else
			{
				isRotating = false;
				time = 0.0f;
				yield break;
			}
		}
	}
}
```

LightPulse.cs

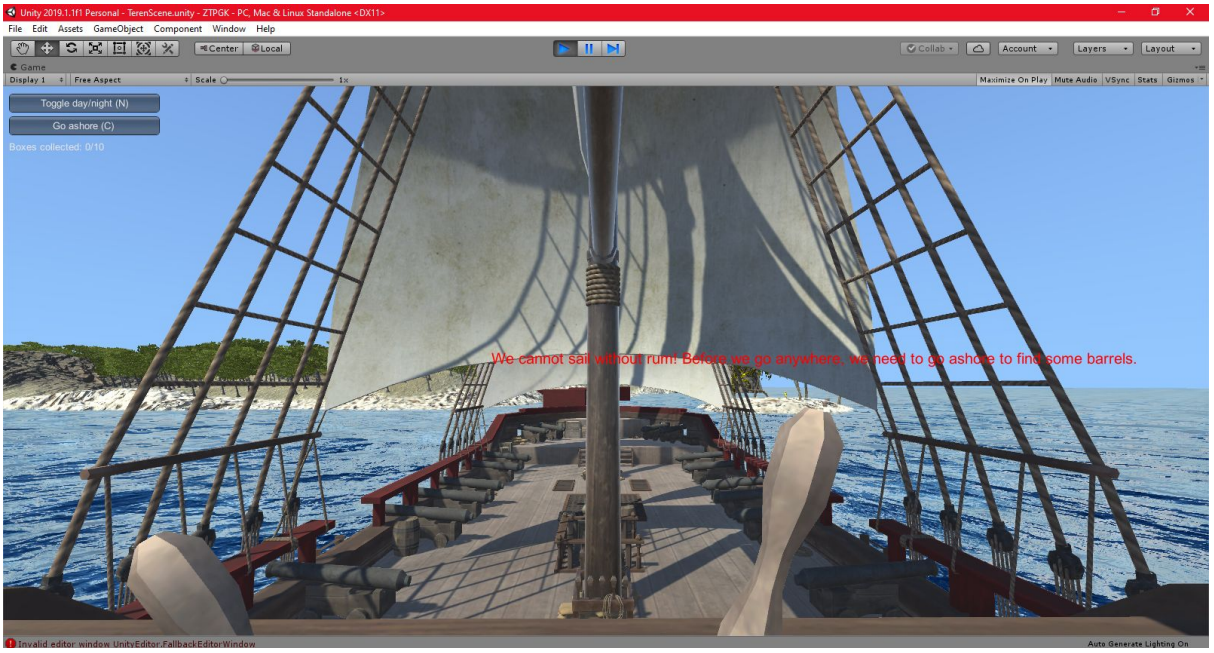
```
using UnityEngine;

public class LightPulse : MonoBehaviour
{
	private Light light;

	public float minIntensity;
	public float maxIntensity;

	private float targetIntensity;
	private float currentIntensity;

	private void Start()
	{
		light = GetComponent<Light>();
		targetIntensity = maxIntensity;
	}

	private void Update()
	{
		currentIntensity = Mathf.MoveTowards(light.intensity,
targetIntensity, 0.1f);

		if (currentIntensity >= maxIntensity)
		{
			targetIntensity = minIntensity;
			currentIntensity = maxIntensity;
		}
		else if (currentIntensity <= minIntensity)
		{
			targetIntensity = maxIntensity;
			currentIntensity = minIntensity;
		}

		light.intensity = currentIntensity;
```

```
        }
}
```

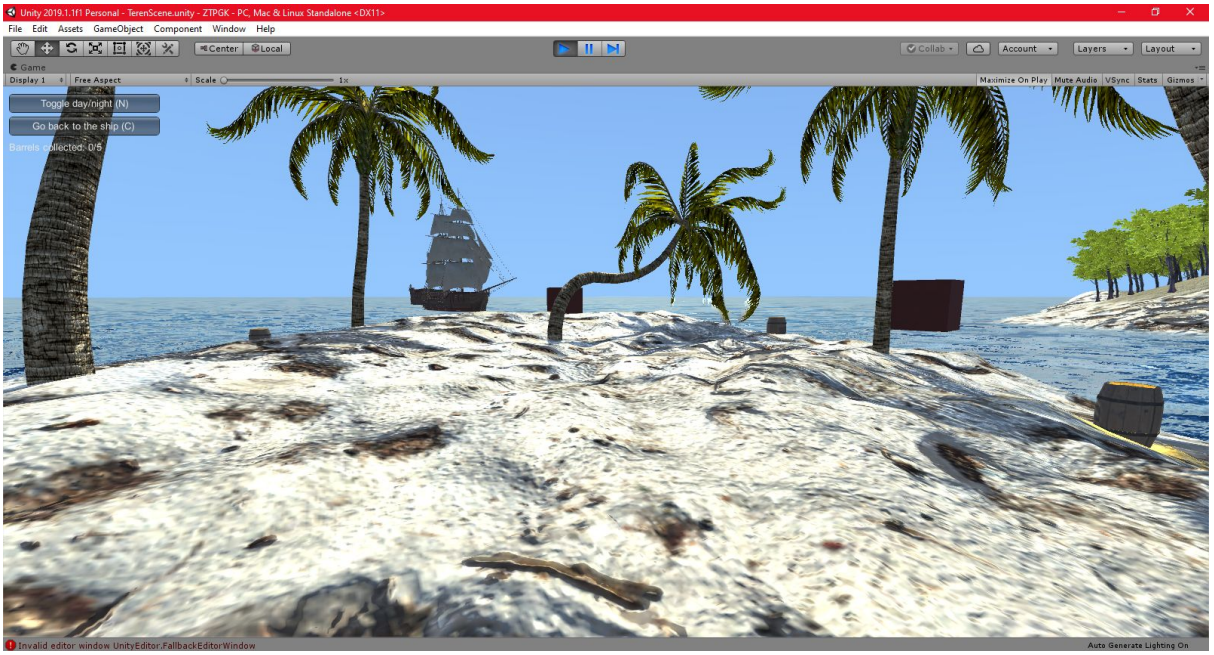
Wszystkie rodzaje drzew obecne na wyspach posiadają komponenty typu Collider i uniemożliwiają graczowi przenikanie przez nie.

3. Specyfikacja zewnętrzna

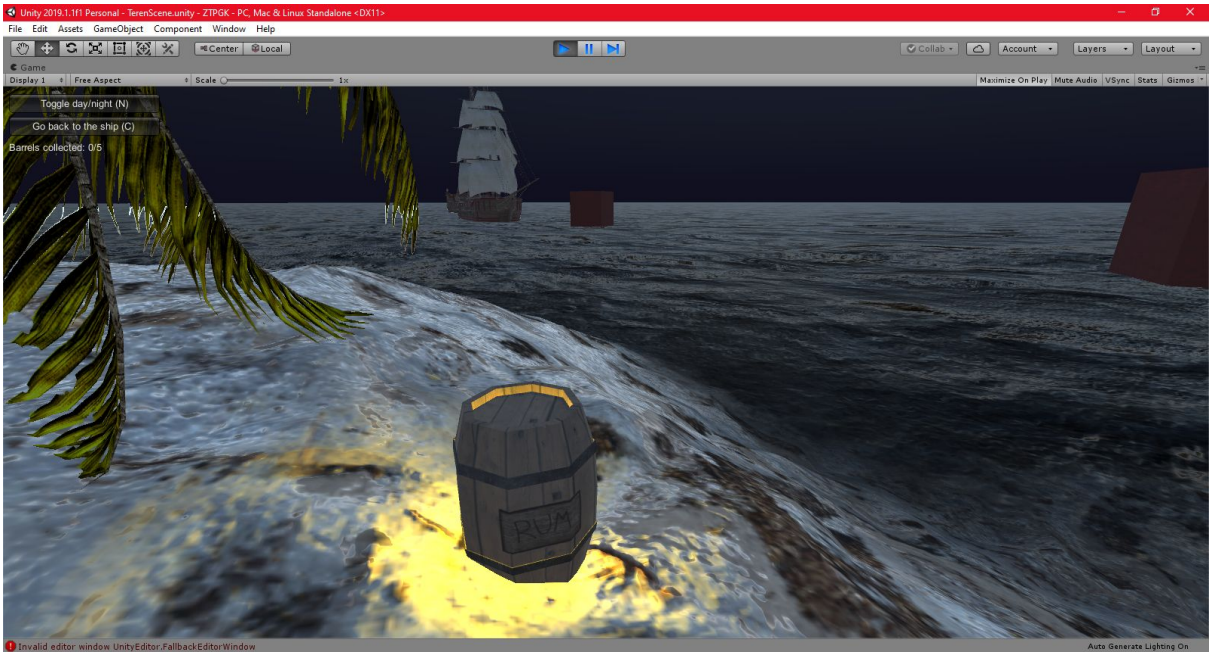
Po rozpoczęciu rozgrywki gracz znajduje się na statku. W lewym górnym rogu znajdują się informacje dotyczące klawiszy umożliwiających przełączanie między lądem, a morzem oraz dniem i nocą. Poniżej widać stan przebiegu misji morskiej. Natomiast na środku wyświetlona jest informacja co należy zrobić.Gracz może rozglądać się za pomocą myszki.



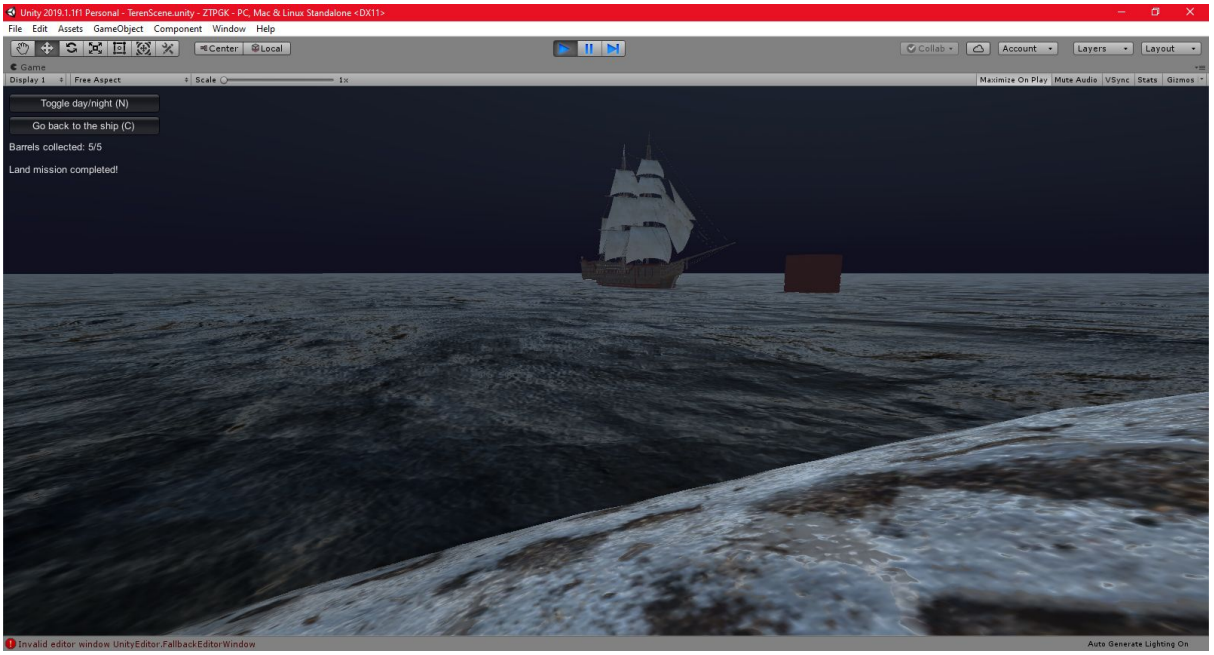
Po naciśnięciu klawisza C, gracz schodzi na ląd i ma możliwość sterowania postacią za pomocą klawiszy WASD oraz rozglądania się za pomocą myszki.



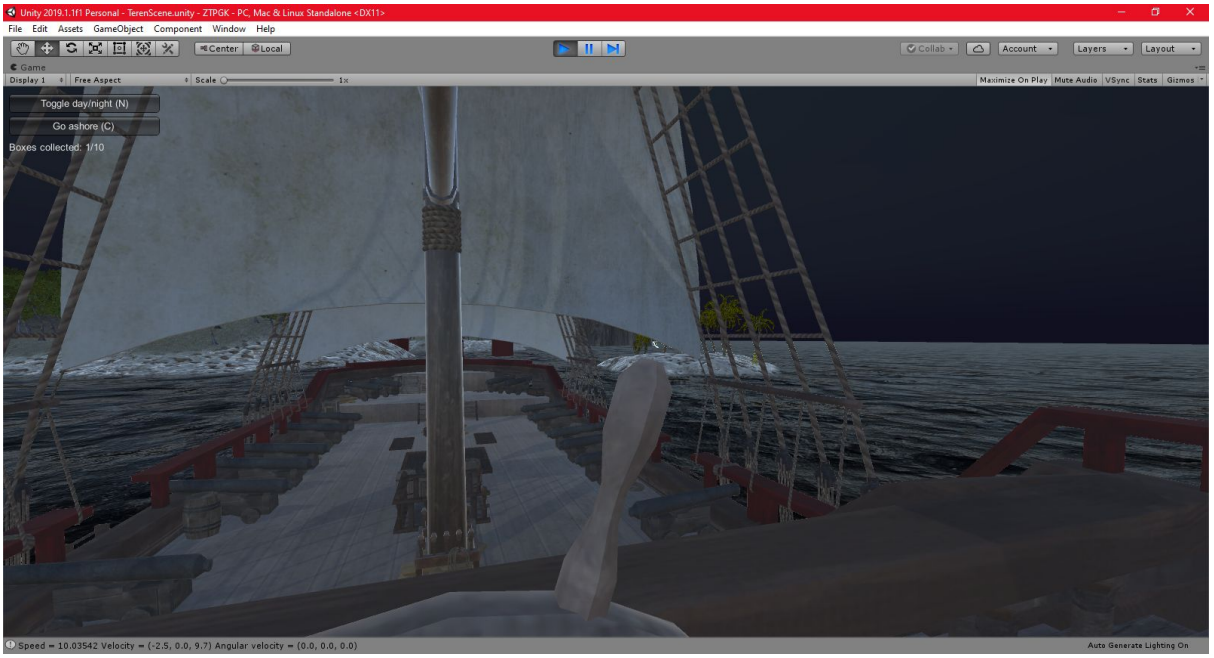
Gracz ma za zadanie zebrać beczki rozrzucone na wyspie. W tym celu musi podejść do nich wystarczająco blisko i nacisnąć klawisz E na klawiaturze. Zebranie beczki poskutkuje zwiększeniem licznika posiadanych beczek oraz usunięciem jej z wyspy.

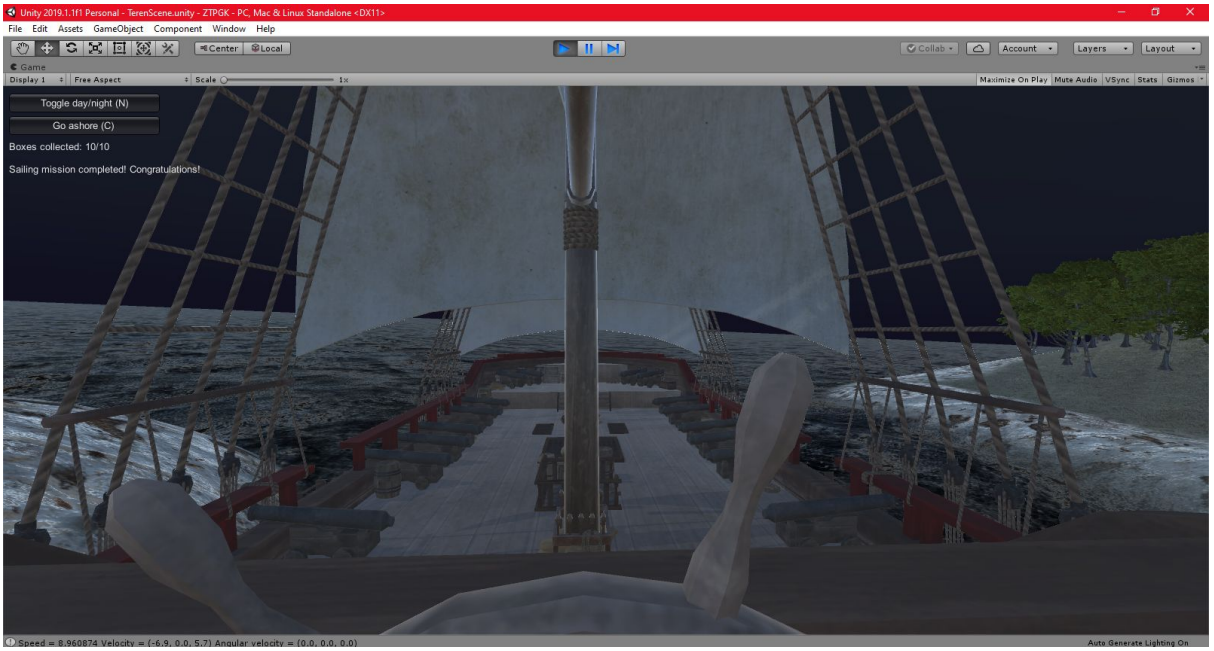


Po zebraniu wszystkich beczek gracz otrzymuje informację, że wykonał misję. Następnie powinien wrócić na statek używając klawisza C.



Po powrocie gracz ma możliwość sterowania statkiem również za pomocą klawiszy WASD. Jego nowym zadaniem jest płynięcie trasą wyznaczoną przez tajemnicze skrzynki unoszące się nad wodą i zbieranie ich. Łącznie musi ich zebrać 10. Po przepłynięciu trasy i zebraniu wszystkich pudełek gracz otrzyma informację o zakończeniu misji.





4. Testy

Podczas testów wyraźnie widoczna jest obniżona wydajność. Najbardziej odczuwalna podczas rozglądania się. Spowodowane jest to najprawdopodobniej ustawieniami obcinania renderowania kamery oraz ustawieniami szczegółowości drzew w konfiguracji terenu. Skraj terenu nie został zabezpieczony przez co możliwe jest wypłynięcie, a nawet wyjście poza teren przeznaczony do rozgrywki. Podczas testów wykryto również błędy związane ze zmianą rotacji statku w wyniku kolizji z terenem. Pierwsze dwa problemy można rozwiązać poprzez zoptymalizowanie ustawień zasięgu renderowania kamery oraz obniżenie poziomu szczegółowości drzew. Skraj mapy można zabezpieczyć dodając komponenty BoxCollider na krawędziach. Problem ze zmianą rotacji wskutek kolizji wystąpił jednorazowo, a doraźnym rozwiązaniem było zablokowanie rotacji w osi X oraz Z w ustawieniach komponentu RigidBody.

5. Podsumowanie

Problematyka projektu nie była bardzo złożona, ale stanowiła interesujące wyzwanie implementacyjne, a dzięki silnikowi Unity3D pozwoliła uzyskać dobre efekty wizualne. Uważam, że projekt pozwolił na usystematyzowanie oraz wykorzystanie w praktyce wiedzy zdobytej podczas realizacji laboratoriów z terenu oraz fizyki. Narzędzia do budowy terenu okazały się bardzo proste w obsłudze przez co nawet początkujący użytkownicy nie powinni mieć problemu z ich użyciem. Stosunkowo niedużym nakładem czasu można zbudować naprawdę interesujące poziomy, a większe doświadczenie w korzystaniu z tego narzędzia z pewnością mogłoby przynieść znacznie lepsze rezultaty. Elementy fizyki stanowią potężne narzędzie, których wykorzystanie skutkuje na przykład podniesieniem poziomu realizmu rozgrywki chociażby ze względu na fakt, że obiekty mogą ze sobą

kolidować. Ponadto umożliwiają stosowanie parametrów i operacji fizycznych takich jak nadanie masy, tarcia, siły czy momentu siły obiektom. Dużym atutem tego typu projektów jest możliwość ich dalszego rozwijania. W ramach kontynuacji prac nad projektem można by poprawić sposób obsługi misji, dodać więcej elementów UI czy też wprowadzić system uszkodzeń statku. Na dalszym etapie prac, kluczowa była by również optymalizacja.