KAUNAS UNIVERSITY OF TECHNOLOGY

FACULTY OF INFORMATICS

T120B166 Development of Computer Games and Interactive Applications

The CQB Project

Group, Name and Surname:
Unspoken Games
Lukas Navašinskas,
IFF-9/8

Kaunas, 2022

Tables of Contents

Work Distribution Table:

Name/Surname	***Description of game development part***
Lukas Navašinskas	Designer, Developer, Tester

Description of Your Game

Description of Your Game.

1. Type: Unity 2D project.
    - Render Pipeline: Universal Render Pipeline
2. What type is your game?
    - Top down strategy shooter game
3. What genre is your game?
    - Roguelite or Action-Adventure
4. Platforms (mobile, PC or both?)
    - PC
5. Scenario Description:
    - You're a next-gen AI robot, your life's purpose is to do what you're asked to do. Soon after your introduction to the world you begin to realize that things you're doing are masked by an illusion, some sort of filter, which masks the horrible reality. After few missions you understand that you're being controlled by something to do bad things. Further into game you get more self-awareness and self-control, as well as abilities, which will help you to finally meet your creators and control your freedom.

Laboratory work #1

List of tasks (main functionality of your project)

1. Create and setup 2D Unity project
2. Create a simple 2D map for testing purposes
3. Implement new Input system controller
4. Implement Player's movement controllers
5. Implement Player's shooting controller
6. Implement bullet controller
7. Implement Camera controller
8. Implement behavior trees for AI development
9. Implement A* pathfinding
10. Implement hash tree optimization of A* pathfinding

Solution

Task #1. *Create and setup 2D Unity project*

New project was created using Unity 2021.2.9f1 2D template. Imported new Input system package which overrides old input event system. BitGem Texture pack was imported from Unity Asset store personal account. Jira project was created alongside with epics and tasks

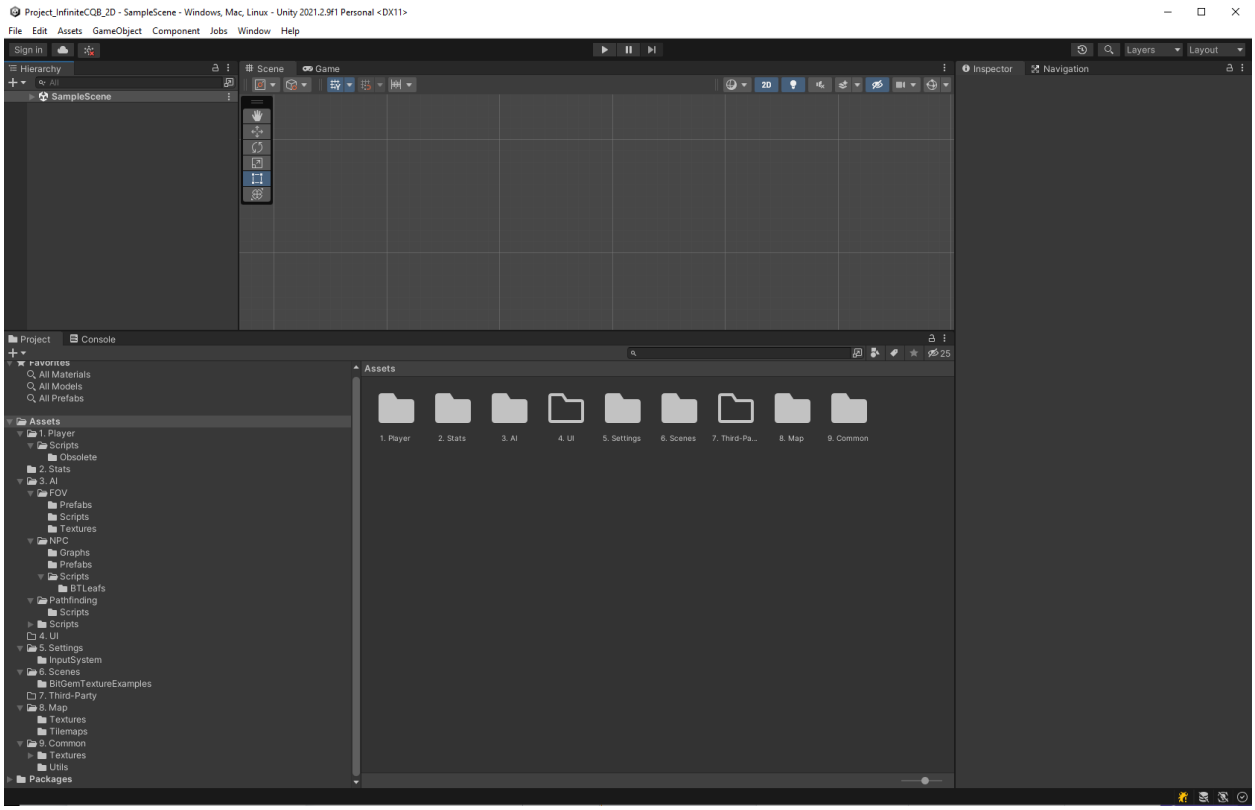

Figure 1. Created project with implemented file structure

Task #2. *Create a simple 2D map for testing purposes*

Created a simple Tilemap which will be used to test out tools and controllers implemented in future. Tilemap uses orange and red color for objects that will have colliders, whilst other textures and colors are used for background.

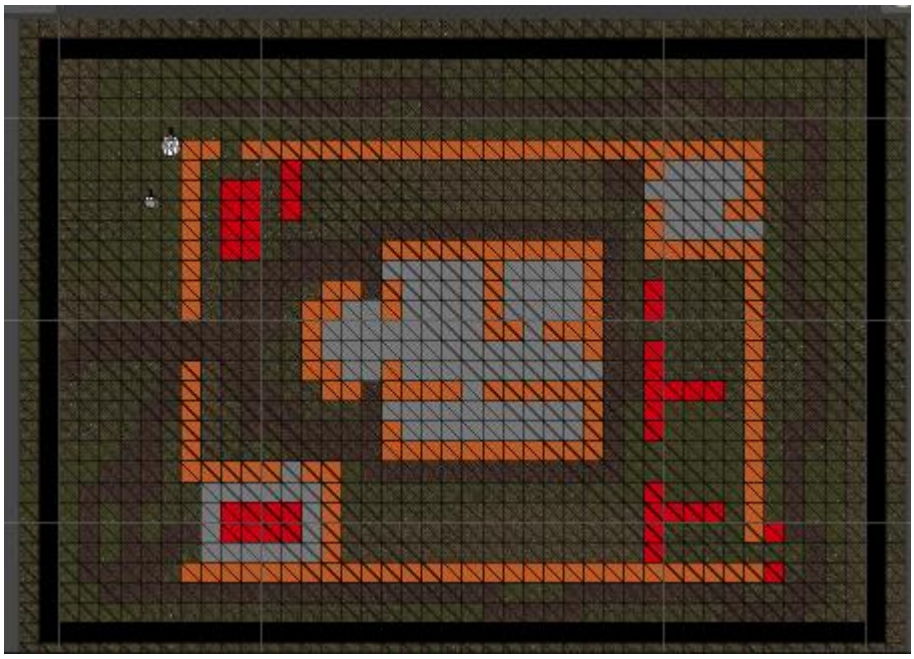

Figure 2. 2D testing map

Task #3. *Implement new Input system controller*

Input system should be easily accessible for multiple controllers so I've used delegates. Delegates are being invoked once certain inputs are detected. Controllers which are subscribed to the delegates can now have easy access to the player inputs.

```
using System.Collections;
using UnityEngine;
using UnityEngine.InputSystem;
using UnityEngine.InputSystem.Interactions;

public class PlayerInputController : MonoBehaviour
{
    /*public InputMaster controls;

    void Awake() {
        controls = new InputMaster();
        controls.Player.Movement.performed += ctx => Move(ctx.ReadValue<Vector2>());
    } --- Consider changing into this*/

    public delegate void Input_OnMoveDelegate(Vector2 movementDirection);
    public static Input_OnMoveDelegate input_OnMoveDelegate;

    public delegate void Input_OnMoveTowardsMouseDelegate(bool isMovingTowardsMouse);
    public static Input_OnMoveTowardsMouseDelegate input_OnMoveTowardsMouseDelegate;

    public delegate void Input_OnSprintDelegate(bool isSprinting);
    public static Input_OnSprintDelegate input_OnSprintDelegate;


    public delegate void Input_OnSpinCameraDelegate(int direction);
```

```
    public static Input_OnSpinCameraDelegate input_OnSpinCameraDelegate;

    public delegate void Input_OnFireDelegate();
    public static Input_OnFireDelegate input_OnFireDelegate;

    public void OnMove(InputAction.CallbackContext context)
    {
        Vector2 movementDirection = context.ReadValue<Vector2>();
        input_OnMoveDelegate(movementDirection);
    }

    public void OnMoveTowardsMouse(InputAction.CallbackContext context)
    {
        if (context.interaction is HoldInteraction && context.canceled)
        {
            input_OnMoveTowardsMouseDelegate(false);
            return;
        }
        if (!context.performed)
            return;
        if (context.interaction is MultiTapInteraction)
            input_OnMoveTowardsMouseDelegate(true);
    }

    public void OnSprint(InputAction.CallbackContext context)
    {
        input_OnSprintDelegate(context.started || context.performed);
    }

    public void OnTurnCamera(InputAction.CallbackContext context)
    {
        if (!context.performed || context.canceled)
            return;
        int direction = Mathf.RoundToInt(context.ReadValue<float>());
        input_OnSpinCameraDelegate(direction);
    }
    public void OnFire(InputAction.CallbackContext context)
    {
        if (InputActionPhase.Started == context.phase)
        {
            input_OnFireDelegate();
        }
    }

}
```

Table 1. Input system controller

Task #3. *Implement new Input system controller*

Input system should be easily accessible for multiple controllers so I've used delegates. Delegates are being invoked once certain inputs are detected. Controllers which are subscribed to the delegates can now have easy access to the player inputs.

```
using System.Collections;
using UnityEngine;
using UnityEngine.InputSystem;
using UnityEngine.InputSystem.Interactions;

public class PlayerInputController : MonoBehaviour
{
    /*public InputMaster controls;

    void Awake() {
        controls = new InputMaster();
        controls.Player.Movement.performed += ctx => Move(ctx.ReadValue<Vector2>());
    } --- Consider changing into this*/

    public delegate void Input_OnMoveDelegate(Vector2 movementDirection);
    public static Input_OnMoveDelegate input_OnMoveDelegate;

    public delegate void Input_OnMoveTowardsMouseDelegate(bool isMovingTowardsMouse);
    public static Input_OnMoveTowardsMouseDelegate input_OnMoveTowardsMouseDelegate;

    public delegate void Input_OnSprintDelegate(bool isSprinting);
    public static Input_OnSprintDelegate input_OnSprintDelegate;


    public delegate void Input_OnSpinCameraDelegate(int direction);
    public static Input_OnSpinCameraDelegate input_OnSpinCameraDelegate;

    public delegate void Input_OnFireDelegate();
    public static Input_OnFireDelegate input_OnFireDelegate;

    public void OnMove(InputAction.CallbackContext context)
    {
        Vector2 movementDirection = context.ReadValue<Vector2>();
        input_OnMoveDelegate(movementDirection);
    }

    public void OnMoveTowardsMouse(InputAction.CallbackContext context)
    {
        if (context.interaction is HoldInteraction && context.canceled)
        {
            input_OnMoveTowardsMouseDelegate(false);
            return;
        }
        if (!context.performed)
            return;
        if (context.interaction is MultiTapInteraction)
            input_OnMoveTowardsMouseDelegate(true);
    }

    public void OnSprint(InputAction.CallbackContext context)
    {
        input_OnSprintDelegate(context.started || context.performed);
    }
```

```
    public void OnTurnCamera(InputAction.CallbackContext context)
    {
        if (!context.performed || context.canceled)
            return;
        int direction = Mathf.RoundToInt(context.ReadValue<float>());
        input_OnSpinCameraDelegate(direction);
    }
    public void OnFire(InputAction.CallbackContext context)
    {
        if (InputActionPhase.Started == context.phase)
        {
            input_OnFireDelegate();
        }
    }

}
```

Table 2. Input system controller code

Task #4 Implement Player's movement controllers

Crated a customizable player movement controller. Controller is intended to be modifiable and should serve two main purposes:

- Subscribe to inputs
- Move and rotate the character

```
using System.Collections;
using UnityEngine;


    public class PlayerMovementController : MonoBehaviour
    {
        public float walkSpeed = 1.5f;
        public float sprintSpeed = 5f;//not using multiplier because, if player walks
slowly with a controller, and clicks sprint - see, see the problem? DO YOU SEE IT?
        //public float backwardsSpeedMultiplier = 0.75f;
        public float walkRotationSmoothTime = 0.5f;
        public float sprintRotationSmoothTime = 0.2f;
        //public float moveTowardsMouseDampDistance = 1f; //When moving towards mouse,
and distance to mouse from character is small - movement speed has to decrease. This is
the distance from which the speed decreasing should occur
        //public float moveTowardsMouseDeadzone = 0.4f;
        public float turnDampTime = 0.1f;

        internal Vector2 _movementDirection;
        internal bool _isSprinting = false;
        internal bool _isMovingTowardsMouse = false;
        internal float charactersRotationDelta;//used to animate character's rotation


        private CameraController _cameraController;
        private PlayerController _playerController;

        void Awake()
        {
            SubscribeToInputSystem();
            GetDependencies();
        }

        private void SubscribeToInputSystem()
        {
```

```
            PlayerInputController.input_OnMoveDelegate += OnMove;
            PlayerInputController.input_OnSprintDelegate += OnSprint;
        }
        private void GetDependencies()
        {
            _cameraController = GetComponent<CameraController>();
            if (_cameraController == null)
                Debug.LogError("CameraController missing");

            _playerController = GetComponent<PlayerController>();
            if (_playerController == null)
                Debug.LogError("PlayerController missing");
        }

        private void Update()
        {
            MoveCharacter();
            RotateCharacter();
        }

        public void OnMove(Vector2 movementDirection)
        {
            _movementDirection = movementDirection;
        }
        public void OnSprint(bool isSprinting)
        {
            _isSprinting = isSprinting;
        }

        private void MoveCharacter()
        {
            float speed = _isSprinting ? sprintSpeed : walkSpeed;
            //speed *= _movementDirection.y <= 0 ? backwardsSpeedMultiplier : 1;//if
moving backwards - multiply the speed by backwardsSpeedMultiplier to slow character
down

            Vector3 movementDirection = new Vector3(_movementDirection.x,
_movementDirection.y, 0f);
            movementDirection.Normalize();
            //movementDirection = Quaternion.Euler(0f, 0f,
(int)_cameraController.CameraDirection * 90f) * movementDirection;//rotate movement
vector towards relative screen rotation
            transform.Translate(movementDirection * speed * Time.deltaTime,
Space.World);

        }

        private void RotateCharacter()
        {
            Vector3 direction = _playerController.mouseTarget - transform.position;
            float rotation = Mathf.Atan2(direction.y, direction.x) * Mathf.Rad2Deg -
90f;
            float speedModifier = 1f;

            float rotationSmoothTime = _isSprinting ? sprintRotationSmoothTime :
walkRotationSmoothTime;
            transform.rotation = Quaternion.Lerp(transform.rotation,
Quaternion.Euler(0f, 0f, rotation), rotationSmoothTime * speedModifier);

            float currentAngle = transform.rotation.eulerAngles.z;
            charactersRotationDelta = rotation < 0 ? rotation + 360 - currentAngle :
rotation - currentAngle;

        }
    }
```

Table 4. Player movement controller code

Task #5. *Implement universal shooting controller*

Implemented a shooting controller intended to be used universally for both AI and Player. Shooting controller avoids constantly instantiating bullet prefabs (which is quite expensive) and instead of it has a collection of bullets already instantiated and deactivated. Once bullet is fired, the bullet is activated once more and force velocity of the bullet is changed.

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.InputSystem;

public class ShootingController : MonoBehaviour
{
    [Header("GameObject references")]
    public Transform firePoint;
    public GameObject bullet;
    // Start is called before the first frame update

    [Header("Optimization settings")]
    public int bulletPoolSize = 15;// maximum count of pooled bullet objects
    private List<GameObject> _bulletPool; // currently instantiated bullets

    [Header("Weapon Settings")]
    //public float rateOfFire = 0.7f; //M1911 pistol Rate of fire is 85 rounds/min,
thus 60s/85 = 0.7s delay
    public float bulletSpeed = 10f;
    public float fireRate = 0.3f;
    private float shootTimer;
    private bool isSprinting = false;
    private bool _isWeaponDrawn = false;

    void Start()
    {
        _bulletPool = new List<GameObject>();
        for (int i = 0; i < bulletPoolSize; i++)
        {
            GameObject go = Instantiate(bullet);
            go.SetActive(false);
            _bulletPool.Add(go);
        }
        shootTimer = 0f;
    }

    // Update is called once per frame
    void Update()
    {
        shootTimer += Time.deltaTime;
    }

    public void OnSprint(InputAction.CallbackContext context)
    {
        isSprinting = context.started || context.performed;
    }

    public void Fire()
    {
        if(shootTimer > fireRate)
```

```
        {
            shootTimer = 0f;

            for (int i = 0; i < _bulletPool.Count; i++)
            {
                if (!_bulletPool[i].activeInHierarchy)
                {
                    _bulletPool[i].transform.position = firePoint.position;
                    _bulletPool[i].transform.rotation = firePoint.rotation;
                    _bulletPool[i].SetActive(true);

                    //if (!isSprinting)
                    //{
                    //    ShootTowardMouse(i);
                    //    break;
                    //}

                    _bulletPool[i].GetComponent<Rigidbody2D>().velocity = firePoint.up
* bulletSpeed;
                    break;
                }
            }
        }


    }

    public void ShootTowardMouse(int bulletPoolIndex)
    {
        Ray ray = Camera.main.ScreenPointToRay(Mouse.current.position.ReadValue());//--
-THIS COULD BE REUSED FROM PlayerController!!!!!!!!
        Plane plane = new Plane(Vector3.forward, Vector3.zero);
        float distance;
        if (plane.Raycast(ray, out distance))
        {
            Vector3 firepointPos = firePoint.position;

            Vector3 targetPos = ray.GetPoint(distance);
            targetPos.z = firepointPos.z;
            Vector3 direction = targetPos - firepointPos;
            direction.Normalize();
            _bulletPool[bulletPoolIndex].GetComponent<Rigidbody2D>().velocity =
direction * bulletSpeed;
        }
    }

    public void Fire(Transform target)
    {
        if (shootTimer > fireRate)
        {
            shootTimer = 0f;

            for (int i = 0; i < _bulletPool.Count; i++)
            {
                if (!_bulletPool[i].activeInHierarchy)
                {
                    _bulletPool[i].transform.position = firePoint.position;
                    _bulletPool[i].transform.rotation = firePoint.rotation;
                    _bulletPool[i].SetActive(true);

                    if (!isSprinting)
                    {
                        ShootTarget(i, target);
                        break;
                    }
```

```
                    _bulletPool[i].GetComponent<Rigidbody2D>().velocity = firePoint.up
* bulletSpeed;
                    break;
                }
            }
        }
    }

    public void ShootTarget(int bulletPoolIndex, Transform target)//used mostly for
NPCs
    {
        Ray ray = Camera.main.ScreenPointToRay(Mouse.current.position.ReadValue());//--
-THIS COULD BE REUSED FROM PlayerController!!!!!!!!
        Plane plane = new Plane(Vector3.forward, Vector3.zero);
        float distance;
        if (plane.Raycast(ray, out distance))
        {
            Vector3 targetPos = target.position;
            Vector3 direction = targetPos - firePoint.position;
            direction.Normalize();
            _bulletPool[bulletPoolIndex].GetComponent<Rigidbody2D>().velocity =
direction * bulletSpeed;
        }
    }
}
```

Table 5. Shooting controller code

Task #6. *Implement bullet controller*

Implemented a bullet controller, which handles the bullets collisions and uses raycasting to detect incoming object collisions. This is necessary when bullet is too fast for the refresh rate of the physics systems. A self-destroy timer is also added to disable bullets after certain period of time.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class BulletController : MonoBehaviour
{
    public TrailRenderer trailRenderer;
    public float damage = 15f;
    public float selfDestroyTime = 2f;
    public float selfDestroyDistance = 0.3f;

    private bool _isActive = false;
    private RaycastHit2D _rayHit;
    //We don't want to destroy and instantiate bullets all the time - we want to enable
and disable them, which is more effective
    private void OnEnable()
    {
        trailRenderer.Clear();
        Invoke("DestroySelf", selfDestroyTime);
        _isActive = true;
    }

    private void OnDisable()
```

```
    {
        CancelInvoke("DestroySelf");
        _isActive = false;
    }

    public void Update()
    {
        if (_isActive)
        {
            Vector3 velocityDir =
gameObject.GetComponent<Rigidbody2D>().velocity;//constantly calculating - really
inefficient since bullet's trajectory doesn't really change does it?
            RaycastHit2D rayHit = Physics2D.Raycast(transform.position,
velocityDir.normalized, selfDestroyDistance);
            if (rayHit)
            {
                OnHit(_rayHit.collider, rayHit);
            }
        }
    }

    public void OnHit(Collider2D other, RaycastHit2D hit)
    {
        Debug.Log("Bullet HIT");
        StatsController statsController;

        if (other.transform.parent != null &&
other.transform.parent.TryGetComponent<StatsController>(out statsController))
        {
            statsController.TakeDamage(damage, hit.point);
        }
        DestroySelf();
    }

    public void OnCollisionEnter2D(Collision2D collision)
    {
        OnHit(collision.collider, _rayHit);
    }



    private void DestroySelf()
    {
        gameObject.SetActive(false);
    }

}
```

Table 6. Title of fragment #3

Task #7. *Implement Camera controller*

Implemented Camera controller which is used to track player and offset camera towards mouse position relative to the player

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
```

```
using UnityEngine;

using Main.Camera.Enums;
using UnityEngine.InputSystem;


    [RequireComponent(typeof(PlayerController))]
    public class CameraController : MonoBehaviour
    {
        [Header("Camera options")]
        public Transform cameraRotationAxis;
        public Transform cameraTarget;
        public float cameraRotationSmoothTime = 0.15f;

        [Header("Camera aim options")]
        public float sensitivity = 0.15f;
        private Rect _screenRect = Rect.zero;

        [Header("Dependencies")]
        public PlayerController playerController;

        //public CameraDirection CameraDirection { get { return _cameraDirection; } }
        //public CameraDirection _cameraDirection = CameraDirection.Forward;

        //private void Awake()
        //{
        //    //PlayerInputController.input_OnSpinCameraDelegate += OnTurnCamera;
        //}

        private void FixedUpdate()
        {
            OffsetCamera();
            //RotateCamera();
        }

        //private void OnTurnCamera(int direction)
        //{
        //    int newDirectionVal = (int)_cameraDirection + direction;
        //    if (newDirectionVal < 0)
        //        newDirectionVal += 4;
        //    else if (newDirectionVal > 3)
        //        newDirectionVal -= 4;
        //    _cameraDirection = (CameraDirection)newDirectionVal; //change the camera
direction
        //}

        private void OffsetCamera()
        {
            _screenRect = new Rect(-Screen.width / 2, -Screen.height / 2, Screen.width,
Screen.height);

            if (_screenRect.Contains(playerController.mouseTarget))//if mouse is still
in the game window
            {
                Vector3 direction = playerController.mouseTarget - transform.position;
                direction.Normalize();
                cameraTarget.position = direction *
playerController.distanceFromCharacterToMouse * sensitivity + transform.position;
            }
        }

        //private void RotateCamera() //rotates the cameraTarget by 'angle' degrees
        //{
```

```
        //    cameraRotationAxis.rotation =
Quaternion.Lerp(cameraRotationAxis.rotation, Quaternion.Euler(0, (int)_cameraDirection
* 90f, 0), cameraRotationSmoothTime);
        //}
    }
```

Table 7. Camera controller code

Task #8. *Implement behavior trees for AI development*

Implemented behaviour tree structure used for development of AI. Behaviour trees help with implementing complicated behaviour of NPC's using custom nodes. With the help of free software yED Graph Editor, these nodes can be stacked in many ways which urges code reusability and solid software design principles

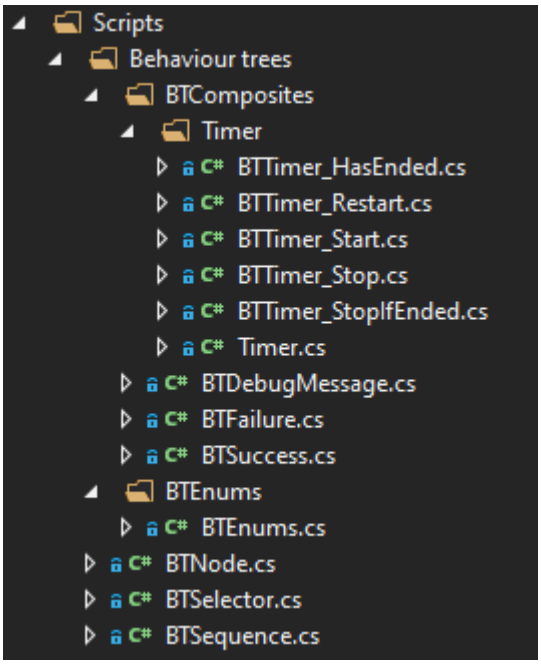

Figure 4. Behavior trees nodes

```
using System.Collections.Generic;
using UnityEngine;
/// <summary>
/// Runs all nodes in a sequence
/// </summary>
public class BTSequence : BTNode
{
    protected List<BTNode> nodes = new List<BTNode>();
    public BTSequence(List<BTNode> nodes)
    {
        this.nodes = nodes;
    }

    public override BTNodeStates Evaluate()
    {
        bool childRunning = false;

        foreach(BTNode node in nodes)
        {
            switch (node.Evaluate())
            {
```

```
                case BTNodeStates.FAILURE:
                    currentNodeState = BTNodeStates.FAILURE;
                    return currentNodeState;

                case BTNodeStates.SUCCESS:
                    continue;

                case BTNodeStates.RUNNING:
                    childRunning = true;
                    continue;

                default:
                    currentNodeState = BTNodeStates.SUCCESS;
                    return currentNodeState;
            }
        }
        currentNodeState = childRunning ? BTNodeStates.RUNNING : BTNodeStates.SUCCESS;
        return currentNodeState;
    }
}
```

```
public class BTFailure : BTNode
{
    public BTFailure()
    {
    }

    public override BTNodeStates Evaluate()
    {
        return BTNodeStates.FAILURE;
    }
}
```

```
public class BTSuccess : BTNode
{
    public BTSuccess()
    {
    }

    public override BTNodeStates Evaluate()
    {
        return BTNodeStates.SUCCESS;
    }
}
```

```
using UnityEngine;

public class BTDebugMessage : BTNode
{
    string message;
    public BTDebugMessage(string message)
    {
        this.message = message;
    }

    public override BTNodeStates Evaluate()
    {
        Debug.Log(message);
        return BTNodeStates.SUCCESS;
    }
}
```

Table 8. Behaviour trees main nodes code

Task #9. *Implement A* pathfinding*

Implemented A* pathfinding which is used to find AI and NPC movement paths. Implementation required scanning the area into a custom grid and finding the walkable areas and non-walkable areas. Using the grid the pathfinding script can now scan trough the tiles in order to find the cheapest path.

```
using UnityEngine;
using System.Collections.Generic;
using Unity.Mathematics;

public class Grid : MonoBehaviour
{
    public LayerMask unwalkableMask;
       public Vector2 gridWorldSize;
       public float nodeRadius = 0.2f;
       public float unitRadius = 0.4f;
       public int MaxSize
    {
        get
        {
                    return gridSizeX* gridSizeY;
        }
    }

       [Header("Debug")]
       public bool shouldDrawGizmos = true;

       Node[,] grid;
       float nodeDiameter;
       int gridSizeX, gridSizeY;

       void Awake()
       {
              nodeDiameter = nodeRadius*2;
              gridSizeX = Mathf.RoundToInt(gridWorldSize.x/nodeDiameter);
              gridSizeY = Mathf.RoundToInt(gridWorldSize.y/nodeDiameter);
              CreateGrid();
       }

       void CreateGrid()
       {
              grid = new Node[gridSizeX,gridSizeY];
              Vector3 worldBottomLeft = transform.position - Vector3.right *
gridWorldSize.x/2 - Vector3.up * gridWorldSize.y/2;

              for (int x = 0; x < gridSizeX; x ++)
              {
                     for (int y = 0; y < gridSizeY; y ++)
                     {
                            Vector3 worldPoint = worldBottomLeft + Vector3.right * (x *
nodeDiameter + nodeRadius) + Vector3.up * (y * nodeDiameter + nodeRadius);
                            bool walkable = !(Physics2D.OverlapCircle(worldPoint,
unitRadius - 0.05f,unwalkableMask));
                            grid[x,y] = new Node(walkable,worldPoint, x,y);
                     }
              }
       }

       public Node NodeFromWorldPoint(Vector3 worldPosition)
       {
              float percentX = (worldPosition.x + gridWorldSize.x/2) / gridWorldSize.x;
              float percentY = (worldPosition.y + gridWorldSize.y/2) / gridWorldSize.y;
```

```
                percentX = Mathf.Clamp01(percentX);
                percentY = Mathf.Clamp01(percentY);

                int x = Mathf.RoundToInt((gridSizeX-1) * percentX);
                int y = Mathf.RoundToInt((gridSizeY-1) * percentY);
                return grid[x,y];
        }

        public List<Node> GetNeighbours(Node node)
    {
                List<Node> neighbours = new List<Node>();

         for (int x = -1; x <= 1; x++){
             for (int y = -1; y <= 1; y++)
                        {
                                if (x == 0 && y == 0)
                                        continue;

                                int checkX = node.gridX + x;
                                int checkY = node.gridY + y;

                                if (checkX >= 0 && checkX < gridSizeX
                                        && checkY >= 0 && checkY < gridSizeY)
                                        neighbours.Add(grid[checkX, checkY]);
                        }
         }

                return neighbours;
    }

        void OnDrawGizmos()
        {
                if (!shouldDrawGizmos)
                        return;

                Gizmos.DrawWireCube(transform.position,new Vector3(gridWorldSize.x,
gridWorldSize.y, 0f));
                if (grid != null) {
                        foreach (Node n in grid) {
                                        Gizmos.color = (n.walkable) ? Color.white :
Color.red;
                                        Gizmos.DrawCube(n.worldPosition, Vector3.one *
(nodeDiameter - .1f));
                        }
                }
        }

        void OnGUI()
        {
                if (GUI.Button(new Rect(10, 70, 100, 30), "Regenerate Grid"))
                        Awake();
        }
}
```

```
using UnityEngine;
using System.Collections;
using Unity.Mathematics;

public class Node : IHeapItem<Node>
{
        public bool walkable;
        public Vector3 worldPosition;
        public Node parent;
        public int gridX;
        public int gridY;
```

```
        public int gCost;
        public int hCost;
        public int fCost { get { return gCost + hCost; } }

        int heapIndex;
        public int HeapIndex
    {
         get
         {
                    return heapIndex;
         }
         set
         {
                    heapIndex = value;
         }
    }
        public Node(bool _walkable, Vector3 _worldPos, int _gridX, int _gridY)
    {
              walkable = _walkable;
              worldPosition = _worldPos;
              gridX = _gridX;
              gridY = _gridY;
        }

    public int CompareTo(Node nodeToCompare)
    {
              int compare = fCost.CompareTo(nodeToCompare.fCost);
              if (compare == 0)
                    compare = hCost.CompareTo(nodeToCompare.hCost);

              return -compare; //negative compare - because the node's value is higher
if the fCost or hCost is lower, thus lower cost == higher value
    }
}
```

Table 3. Grid and Node implementation of A* pathfinding algorithm code

Laboratory work #2

List of tasks

1. Implement shooting effects
2. Implement Night Vision
3. Implement main menu
4. Implement blood effects on bullet collision
5. Implement movement animations for Player
6. Implement movement animations for AI
7. Implement camera shake
8. Implement character's design

Solution

Task #1 Implement shooting effects

Created weapon firing effects using Unity's particle systems.

Figure 5. Weapon firing effect

```
public void Fire()
    {
        if(shootTimer > fireRate)
        {
            shootTimer = 0f;

            for (int i = 0; i < _bulletPool.Count; i++)
            {
                if (!_bulletPool[i].activeInHierarchy)
                {
                    if (isPlayer)
                        cinemachineImpulseSource.GenerateImpulse();

                    particleSystem.Play();

                    _bulletPool[i].transform.position = firePoint.position;
                    _bulletPool[i].transform.rotation = firePoint.rotation;
                    _bulletPool[i].SetActive(true);

                    //if (!isSprinting)
                    //{
                    //    ShootTowardMouse(i);
```

```
                    //    break;
                    //}

                    _bulletPool[i].GetComponent<Rigidbody2D>().velocity = firePoint.up
* bulletSpeed;
                    break;
                }
            }
        }
```

Table 5. Particle system player code

Task #2 Implement Night Vision

- Created a night vision effect using Unity's Volumes and overrides:

Figure 6. Night vision effect

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class GlobalLightningController : MonoBehaviour
{
    public GameObject DisabledLightning;
    public GameObject EnabledLightning;

    private bool _isToggled { get; set; } = false;

    public void Start()
    {
        PlayerInputController.input_ToggleNightVision += ToggleNightVision;
    }

    public void ToggleNightVision()
    {
        _isToggled = !_isToggled;
        DisabledLightning.SetActive(!_isToggled);
```

```
        EnabledLightning.SetActive(_isToggled);
    }
}
```

Table 6. Night vision toggling code

Task #3 Implement main menu

- Created main menu using Unity's Canvas systems along with Unity's Volumes after effects.

Figure 7. Main menu

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class MainMenuController : MonoBehaviour
{
    public void SwitchToCampaignScene()
    {
        SceneManager.LoadScene(1);
    }
}
```

Table 7. Main menu controller script

Task #4 Implement blood effects on bullet collision

- Created visual effects for bullet collisions with enemies and player using Unity's particle systems

Figure 8. Blood splat effect

```
public void TakeDamage(float damage, Vector3 hitDirection)
    {
        isHit = true;
        this.hitDirection = hitDirection;
        //Debug.DrawLine(transform.position, transform.position + hitDirection * 5,
Color.red, 1f);

        health -= damage;
        if (health <= 0)
        {
            deathDelegate();
            return;
        }
        if(isPlayer)
        {
            healthUpdateDelegate();
            screenShakeImpulseSource.GenerateImpulse();
        }
        Debug.DrawRay(transform.position, hitDirection, Color.red, 1f);
        float bloodSplatterRotationAngleTowardsTarget = Mathf.Atan2(hitDirection.y,
hitDirection.x) * Mathf.Rad2Deg - 90f;
        bloodSplatterParticleSystem.transform.rotation = Quaternion.Euler(0f, 0f,
bloodSplatterRotationAngleTowardsTarget);
        bloodSplatterParticleSystem.Play();
    }
```

Table 8. Take damage method that plays blood effects

Task #5 Implement movement animations for Player

- Created player movement animations from scratch using Aseprite tool

Figure 9. Movement animation sheet

```
private void AnimateCharacter()
    {
        float speed = _isSprinting ? sprintSpeed : _isMoving ? walkSpeed : 0f;

        animator.SetFloat("Speed", speed* _movementDirection.magnitude);

    }
```

Table 9. Player movement animation code

Task #6 Implement movement animations for AI

- Modified player's animation controller to fit with AI controllers, so AI NPCs movement would be animated.

```
public class NPCAnimator : MonoBehaviour
{
    public Animator animator;
    public float walkSpeedMultiplier = 1.5f; //Blend tree named "Movement" has sprint
animation, which y position we place here
    public float maximumWalkSpeed = 3f;
    private Vector3 _lastPos;

    void Update()
    {
        Vector3 movementDirection = (_lastPos - transform.position).normalized;
        float speed = movementDirection.magnitude* walkSpeedMultiplier;
        speed = Mathf.Clamp(speed, -maximumWalkSpeed, maximumWalkSpeed);
        animator.SetFloat("Speed", speed);
        _lastPos = transform.position;
    }
}
```

Table 10. Movement animations controller for AI

Task #7 Implement camera shake

- Implemented camera shake which so the game would feel more intuitive. Camera shakes when user shoots weapon or gets hit by a bullet. Camera shake is implemented using Unity's Cinemachine Impulses tools.

```
public void Fire()
    {
        if(shootTimer > fireRate)
        {
            shootTimer = 0f;

            for (int i = 0; i < _bulletPool.Count; i++)
            {
                if (!_bulletPool[i].activeInHierarchy)
                {
                    if (isPlayer)
                        cinemachineImpulseSource.GenerateImpulse();

                    particleSystem.Play();

                    _bulletPool[i].transform.position = firePoint.position;
                    _bulletPool[i].transform.rotation = firePoint.rotation;
                    _bulletPool[i].SetActive(true);

                    //if (!isSprinting)
                    //{
                    //    ShootTowardMouse(i);
                    //    break;
                    //}

                    _bulletPool[i].GetComponent<Rigidbody2D>().velocity = firePoint.up
* bulletSpeed;
                    break;
```

```
                }
            }
        }
    }
```

Table 11. Firing method which calls Impluse sources to send a shockwave

Task #8 Implement character's design

- Created character's design with multiple layers (feet, head, body) using Aseprite tool.

Figure 10. Character's design

Laboratory work #3

List of tasks (main functionality of your project)

1. Shooting SoundFX
2. AI shooting implementation rework
3. AI pathfinding heap optimization
4. AI A* pathfinding same direction movement optimization

Solution

Task #1. *Shooting SoundFX*

Implement sound effect controller for shooting, which can be universally used between AI and Player classes

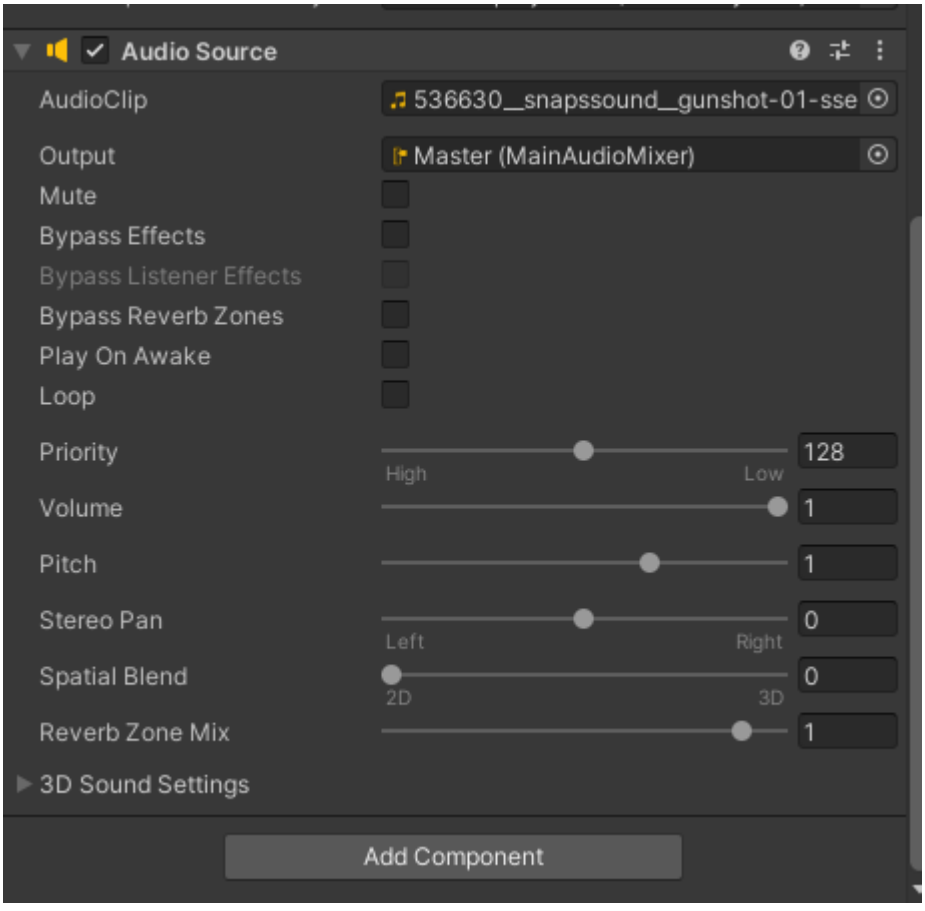

Figure 11. Shoting sound effect Audio Source

```
public void Fire(Transform target)
{
    if (shootTimer > fireRate)
    {
        shootTimer = 0f;

        for (int i = 0; i < _bulletPool.Count; i++)
        {
            if (!_bulletPool[i].activeInHierarchy)
            {
                particleSystem.Play();
                shootingSound.Play();
                _bulletPool[i].transform.position = firePoint.position;
                _bulletPool[i].transform.rotation = firePoint.rotation;
                _bulletPool[i].SetActive(true);

                if (!isSprinting)
                {
                    ShootTarget(i, target);
                    break;
                }

                _bulletPool[i].GetComponent<Rigidbody2D>().velocity = firePoint.up * bulletSpeed;
                break;
            }
        }
    }
}
```

Table 12. Shooting sound effect implementation

Task #2. *Implement hash tree optimization of A* pathfinding*

Since looping through 2D array can get really expensive, which we need to do constantly in A* algorithm in order to find the currently cheapest path, a hashset was implemented in order to keep the path objects collection always sorted.

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;
using System;

public class Pathfinding : MonoBehaviour
{
    private const int DiagonalMoveCost = 14;
       private const int PerpendicularMoveCost = 10;

    PathRequestManager requestManager;
       Grid grid;

       void Awake()
       {
              requestManager = GetComponent<PathRequestManager>();
              grid = GetComponent<Grid>();
       }


       public void StartFindPath(Vector3 startPos, Vector3 targetPos)
       {
              StartCoroutine(FindPath(startPos, targetPos));
       }

       IEnumerator FindPath(Vector3 startPos, Vector3 targetPos)
```

```
	{

		Vector3[] waypoints = new Vector3[0];
		bool pathSuccess = false;

		Node startNode = grid.NodeFromWorldPoint(startPos);
		Node targetNode = grid.NodeFromWorldPoint(targetPos);


		if (startNode.walkable && targetNode.walkable)
		{
			Heap<Node> openSet = new Heap<Node>(grid.MaxSize);
			HashSet<Node> closedSet = new HashSet<Node>();
			openSet.Add(startNode);

			while (openSet.Count > 0)
			{
				Node currentNode = openSet.RemoveFirst();
				closedSet.Add(currentNode);

				if (currentNode == targetNode)
				{
					pathSuccess = true;
					break;
				}

				foreach (Node neighbour in grid.GetNeighbours(currentNode))
				{
					if (!neighbour.walkable ||
closedSet.Contains(neighbour))
					{
						continue;
					}

					int newMovementCostToNeighbour = currentNode.gCost +
GetDistance(currentNode, neighbour);
					if (newMovementCostToNeighbour < neighbour.gCost ||
!openSet.Contains(neighbour))
					{
						neighbour.gCost = newMovementCostToNeighbour;
						neighbour.hCost = GetDistance(neighbour,
targetNode);

						neighbour.parent = currentNode;

						if (!openSet.Contains(neighbour))
							openSet.Add(neighbour);
					}
				}
			}
		}
		yield return null;
		if (pathSuccess)
		{
			waypoints = RetracePath(startNode, targetNode);
		}
		requestManager.FinishedProcessingPath(waypoints, pathSuccess);

	}

	Vector3[] RetracePath(Node startNode, Node endNode)
	{
		List<Node> path = new List<Node>();
		Node currentNode = endNode;

		while (currentNode != startNode)
```

```
                {
                        path.Add(currentNode);
                        currentNode = currentNode.parent;
                }
                Vector3[] waypoints = OptimizePath(path);
                return waypoints;

        }

        Vector3[] OptimizePath(List<Node> path)
        {
                List<Vector3> waypoints = new List<Vector3>();
                Vector2 directionOld = Vector2.zero;
                for (int i = path.Count-1; i > 0; i--)
                {
                        Vector2 directionNew = new Vector2(path[i - 1].gridX -
path[i].gridX, path[i - 1].gridY - path[i].gridY);

                        if (directionNew != directionOld )
                        {
                                waypoints.Add(path[i].worldPosition);
                        }
                        directionOld = directionNew;
                }
                return waypoints.ToArray();
        }

        int GetDistance(Node nodeA, Node nodeB)
        {
                int dstX = Mathf.Abs(nodeA.gridX - nodeB.gridX);
                int dstY = Mathf.Abs(nodeA.gridY - nodeB.gridY);

                if (dstX > dstY)
                        return DiagonalMoveCost * dstY + PerpendicularMoveCost * (dstX -
dstY);
                return DiagonalMoveCost * dstX + PerpendicularMoveCost * (dstY - dstX);
        }
}
```

Task #3. *AI A* pathfinding same direction movement optimization*

When moving the same direction, we don't want keep changing directions to the same one. To avoid repetitive costly actions we could implement a method which checks if NPC is already moving to the same direction

```
1 reference
Vector3[] OptimizePath(List<Node> path)
{
    List<Vector3> waypoints = new List<Vector3>();
    Vector2 directionOld = Vector2.zero;
    for (int i = path.Count-1; i > 0; i--)
    {
        Vector2 directionNew = new Vector2(x: path[i - 1].gridX - path[i].gridX, y: path[i - 1].gridY - path[i].gridY);

        if (directionNew != directionOld )
        {
            waypoints.Add(path[i].worldPosition);
        }
        directionOld = directionNew;
    }
    return waypoints.ToArray();
}
```

Table 12. Title of fragment #3

User's manual (for the Individual work defence)

How to play? *Aenean eu quam gravida, laoreet nisl eu, sagittis quam. Donec sit amet nunc nisi. Sed vel ipsum metus. Nullam accumsan vestibulum ex. Aenean eu quam gravida, laoreet nisl eu, sagittis quam. Donec sit amet nunc nisi. Sed vel ipsum metus. Nullam accumsan vestibulum ex.*

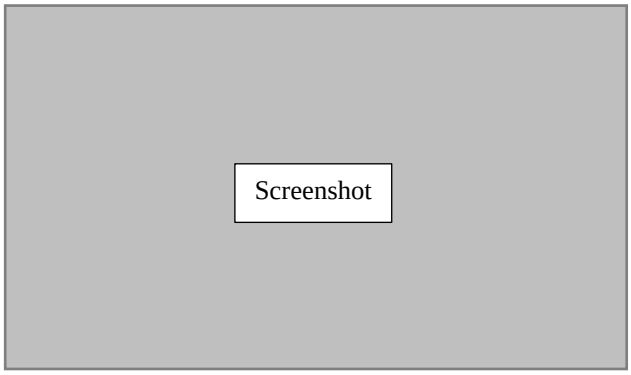

Figure 3. Screenshot #5

Nunc vel enim vel magna interdum dapibus id nec nisl. Suspendisse elit augue, accumsan tempor erat sed, gravida suscipit urna. Duis blandit lacus et finibus finibus. Mauris pretium pharetra orci dictum luctus. Nullam commodo magna a tincidunt malesuada.

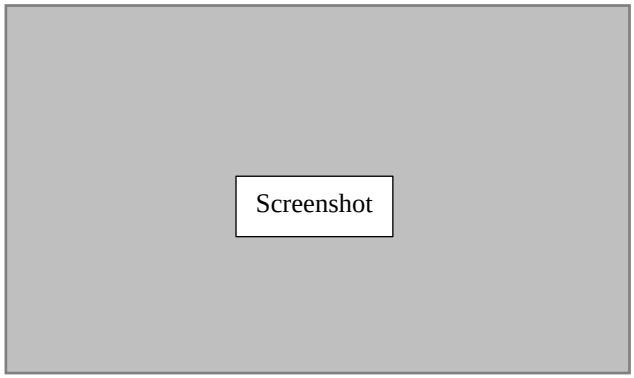

Figure 4. Screenshot #5

Sed sollicitudin justo erat, viverra luctus mi consequat non. Sed ut condimentum libero. Duis rutrum lacus ante, vitae feugiat ex faucibus at. Maecenas pulvinar et augue sed commodo.

Descriptions of the rules of the game. Nunc quis condimentum lacus. Quisque felis neque, ullamcorper vel posuere eget, blandit non neque. Nam in varius erat. Duis molestie sit amet eros vel rhoncus. Nunc quis condimentum lacus. Quisque felis neque, ullamcorper vel posuere eget, blandit non neque. Nam in varius erat. Duis molestie sit amet eros vel rhoncus.

Descriptions of the controls / keys. Donec et lorem vitae ligula bibendum faucibus. Suspendisse interdum quis augue sed luctus. Curabitur ac diam augue. In hac habitasse platea dictumst. Curabitur maximus maximus tortor. Nunc quis condimentum lacus. Quisque felis neque, ullamcorper vel posuere eget, blandit non neque. Nam in varius erat. Duis molestie sit amet eros vel rhoncus.

Literature list

1. Source #1. *Url*
2. Source #2. *Url*
3. ...
4. Source #N. *Url*

ANNEX

All source code is contained in this part.