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

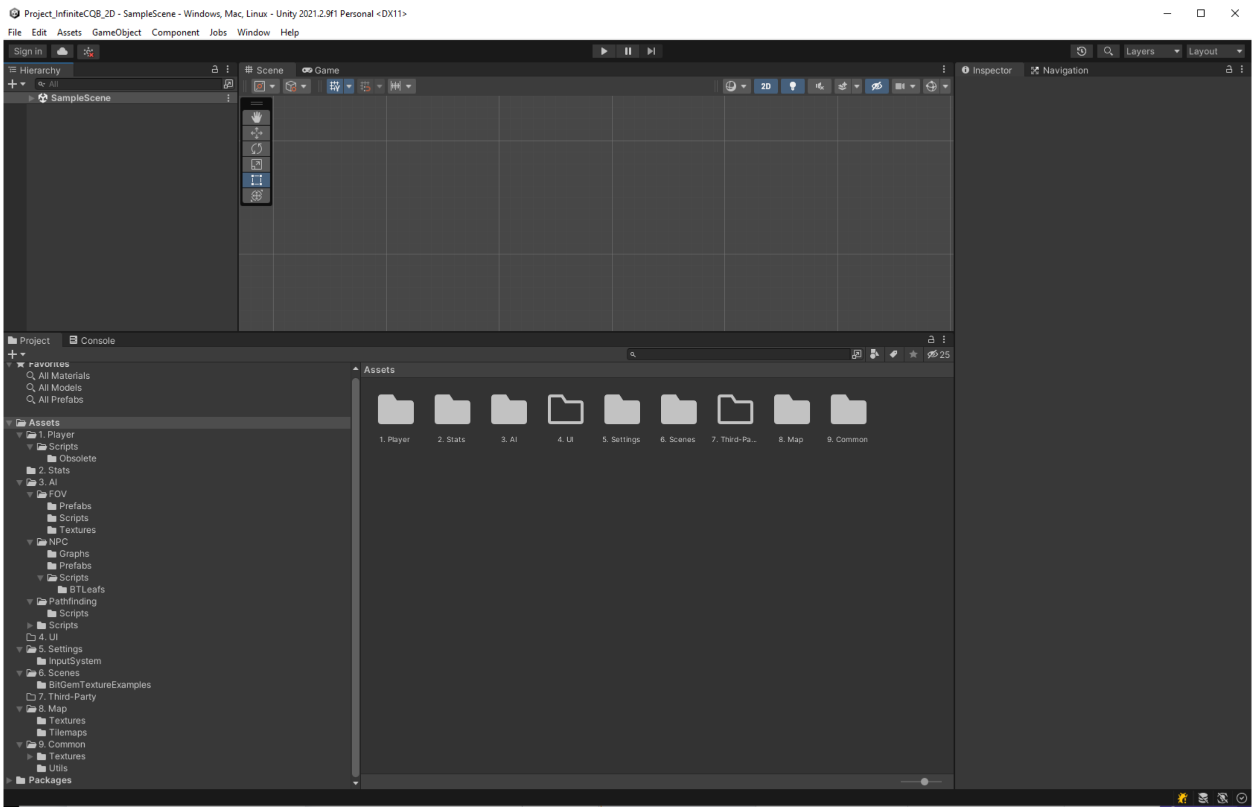


Figure 1. Created project with implemented file structure

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.

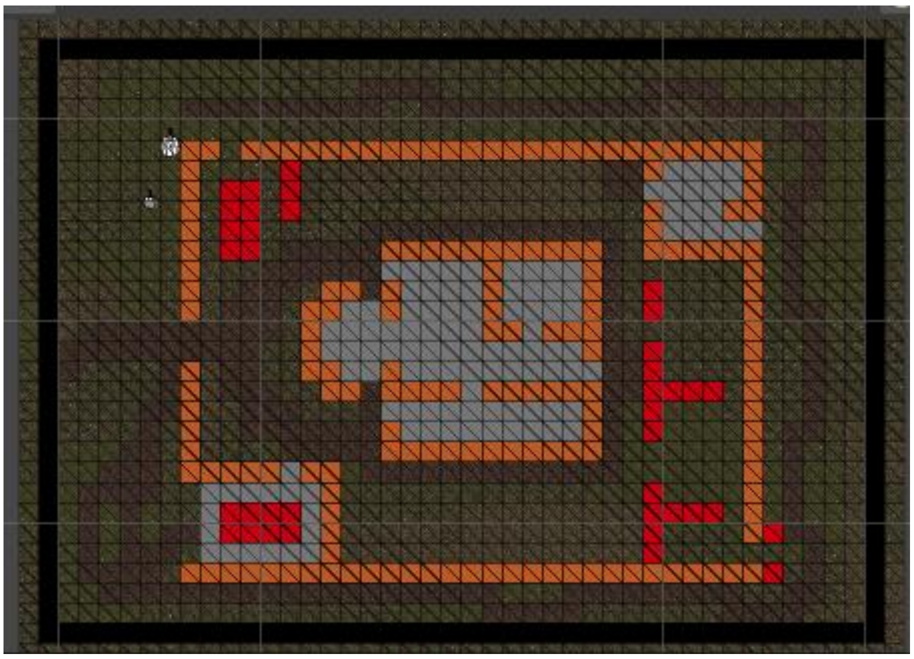


Figure 2. 2D testing map

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

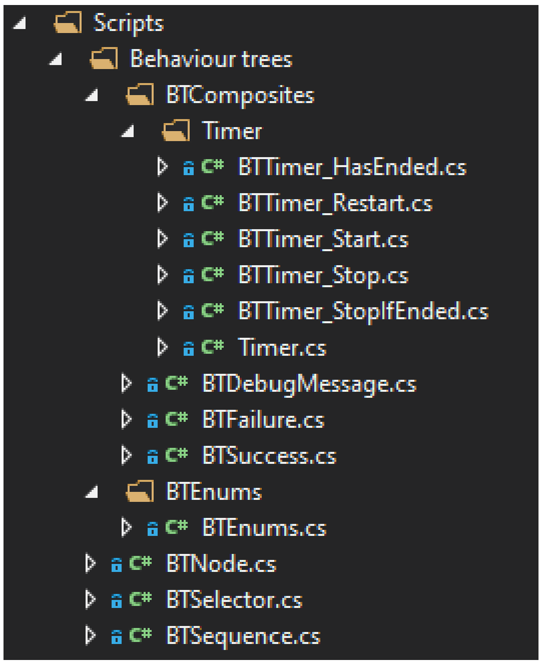
Task #5. *Implement universal shooting controller*

Task #6. *Implement bullet controller*

Task #7. *Implement Camera controller*

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



```
Scripts
    Behaviour trees
        BTComposites
            Timer
                BTTimer_HasEnded.cs
                BTTimer_Restart.cs
                BTTimer_Start.cs
                BTTimer_Stop.cs
                BTTimer_StopIfEnded.cs
                Timer.cs
            BTDebugMessage.cs
            BTFailure.cs
            BTSuccess.cs
        BTEnums
            BTEnums.cs
        BTNode.cs
        BTSelector.cs
        BTSequence.cs
```

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*

Task #10. *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);
        }
}
```

Table 4. A* pathfinding algorithm

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

Task #2 Implement Night Vision

Task #3 Implement main menu

Task #4 Implement blood effects on bullet collision

Task #5 Implement movement animations for Player

Task #6 Implement movement animations for AI

Task #7 Implement camera shake

Task #8 Implement character’s design

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

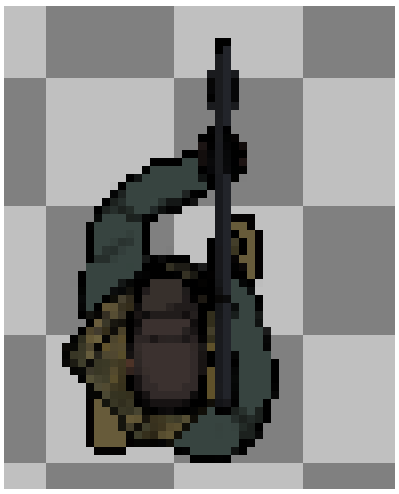


Figure 10. Character’s design

Laboratory work #3

List of tasks (main functionality of your project)

1. Title of Task #1
2. Title of Task #2
3. Title of Task #3
4. …

Solution

Literature list

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

ANNEX

All source code is contained in this part.