Concept Document

THE ROTATING ARKANOID

INTRODUCTION

"The Rotating Arkanoid" is a reaction and intuition trainer. It is also a modern implementation of old arcade game mechanic, which first appeared on the game machines in 1980s. It has a more futuristic and meditational space visual style than its predecessor, while in the same time features the time-tested gameplay mechanics. However, the block moving is implemented in the more complex way than the original one to make it harder for the player to hit the desired block.

UNIQUE SELLING POINT

This game allows you to train your reaction, intuition and dimension-oriented thinking and stimulates the competition between several players.

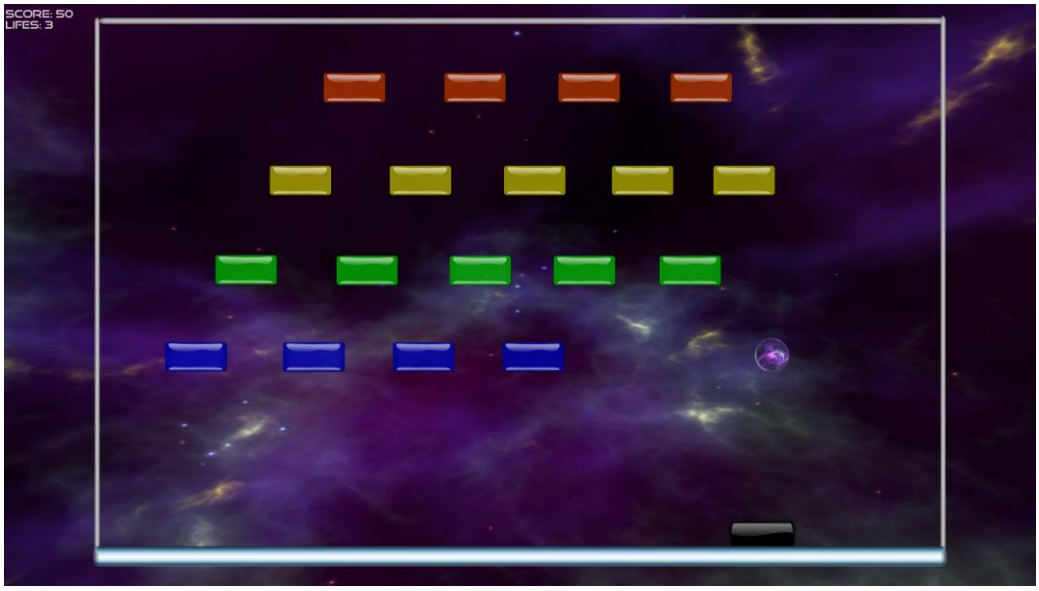
GAMEPLAY

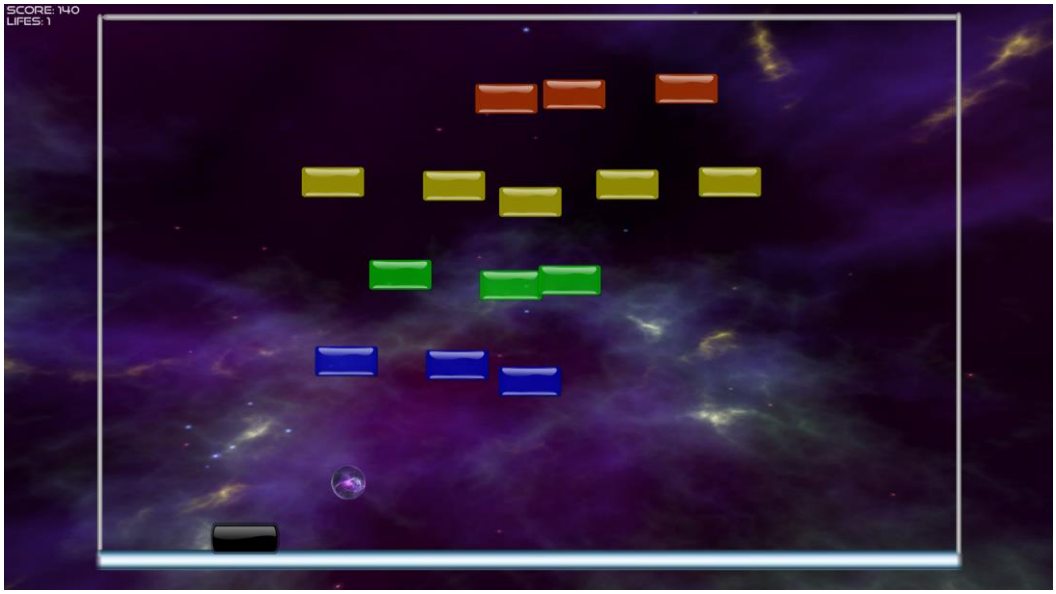
You operate the paddle, which is used to reflect ball in order to prevent it from falling down from the screen (if the ball touches the laser beam – it gets destroyed). The main target is to destroy all blocks that are located in lines on the game field. There are several types of blocks, each of which features its own visual appearance, the durability and the score reward. Blocks are changing their formation in a short period of time, so you need to plan the trajectory of the ball carefully. Moreover, after completing the level, the next one will be rotated by 90 degrees to make the game more excitable and train your dimension-oriented thinking.

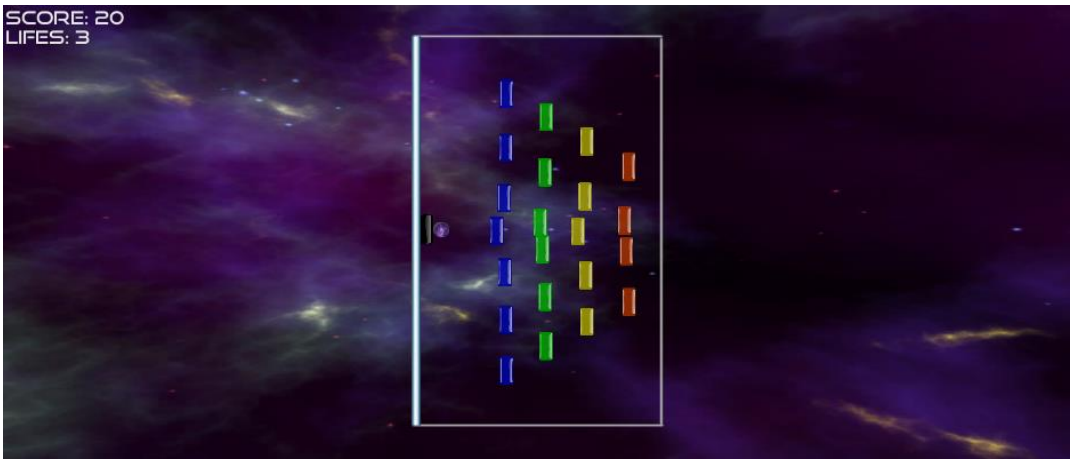
GENRE AND AUDIENCE

Arcade-styled trainer for the whole family.

SCREENSHOTS







MAIN FEATURES OF THE PROJECT

- The game field is rotated by 90 degrees each time you complete a level
- Leaderboards mechanic to bring more competition to the game and make the gameplay more excitable
- The whole-new block movement scripting (blocks group in the center to prevent the ball from passing through their lines)
- Block movement is implemented via Observer programming pattern.

Game Feature	Code Implementation	Comment

Game field rotation	`public class Playerscript : MonoBehaviour`
	`{`
	`void Start()`
	`    {`
	`        loadTime = PlayerPrefs.GetInt("loadTime", 0);`
	`        // Set the camera position`
	`        // Horizontal orientation of game field - either 0 or 180 degrees`
	`        if (loadTime % 2 == 0)`
	`        {`
	`            Camera.main.transform.position = new Vector3(cameraX, cameraY, cameraHorizontalZ);`
	`            Camera.main.orthographicSize = HorizontalOrthographic;`
	`        }`
	`        // Vertical one - 90 or 270 degrees`
	`        else`
	`        {`
	`            Camera.main.transform.position = new Vector3(cameraX, cameraY, cameraVerticalZ);`
	`            Camera.main.orthographicSize = VerticalOrthographic;`
	`        }`
	`        if (loadTime == 0)`
	`        {`
	`            playerPoints = 0;`
	`        }`
	`        else`
	`        {`
	`            // Rotate the game field by the needed amount of times`
	`            for (int i = 0; i < loadTime; i++)`
	`            {`
	`                Camera.main.transform.eulerAngles = new Vector3(0, 0, 90f);`
	`            }`
	`        }`
	`    }`
	`}`
		Used to rotate the player's view by rotating camera to simulate the game field rotation to achieve better performance

The Observer pattern implementation	`interface IObserver`
	`{`
	`    void update();`
	`}`
	
	`interface ISubject`
	`{`
	`    void Subscribe(GameObject go);`
	`    void Unsubscribe(GameObject go);`
	`    void Notify();`
	`}`
		Used to issue movement commands to blocks

Game Feature	Code Implementation	Comment

Issuing the movement command once in 3 seconds	public class Timer : ISubject
	{
	    private List<GameObject> subs = new List<GameObject>();
	    private int secondsToStop;
	    private float t;
	
	    public Timer()
	    {
	        secondsToStop = 3;
	        GameObject[] gos = GameObject.FindGameObjectsWithTag("Block");
	//returns GameObject[]
	        foreach (GameObject g in gos)
	        {
	            Subscribe(g);
	        }
	        t = Time.time;
	    }
	
	    public void CheckTime()
	    {
	        if ((Time.time - t) >= secondsToStop)
	        {
	            t = Time.time;
	            Notify();
	        }
	    }
	    public void Subscribe(GameObject go)
	    {
	        subs.Add(go);
	    }
	
	    public void Unsubscribe(GameObject go)
	    {
	        subs.Remove(go);
	    }
	
	    public void Notify()
	    {
	        foreach (GameObject s in subs)
	        {
	            MonoBehaviour[] list = s.GetComponents<MonoBehaviour>();
	            foreach (MonoBehaviour Mb in list)
	            {
	                if (Mb is IObserver)
	                {
	                    IObserver sh = (IObserver)Mb;
	                    sh.update();
	                }
	            }
	        }
	    }
	}
	public class timerScript : MonoBehaviour
	{
	
	    public Timer Timer;
	    // Use this for initialization
	    void Start()
	    {
	        Timer = new Timer();
	    }
	
	    // Update is called once per frame
		The Timer object is a time controller that acts as the Subject in Observer pattern to send the commands to the blocks

Game Feature	Code Implementation	Comment

	`void Update()`
	`{`
	`Timer.CheckTime();`
	`}`
	
	`void BlockDestroyed(GameObject go)`
	`{`
	`Timer.Unsubscribe(go);`
	`}`
	`}`	

Managing the received command by the block	`public class BlockScript : MonoBehaviour, IObserver`
	`{`
	
	`public int hitsToKill;`
	`public int points;`
	`private int numberOfHits;`
	`public Vector3 pointB;`
	`private Vector3 pointA;`
	`private Vector3 pos;`
	`private bool moved;`
	
	`public void update()`
	`{`
	`if (!moved)`
	`{`
	`MoveFunction(new Vector3(0.0f, transform.position.y - 0.5f, 0.0f));`
	`moved = true;`
	`}`
	`else`
	`{`
	`MoveFunction(pointA);`
	`moved = false;`
	`}`
	`}`
	
	`// Use this for initialization`
	`void Start()`
	`{`
	`numberOfHits = 0;`
	`pointA = gameObject.transform.position;`
	`pos = gameObject.transform.position;`
	`moved = false;`
	`}`
	
	`void MoveFunction(Vector3 endpos)`
	`{`
	`transform.position = Vector3.MoveTowards(transform.position, endpos, 0.5f);`
	`}`
		Each block is subscribed to the Timer object and starts changing its position after receiving the command from Timer