Summary

Generated on:	06/06/2023 - 12.00.18
Coverage date:	06/06/2023 - 11.59.12
Parser:	Cobertura
Assemblies:	2
Classes:	82
Files:	82
Covered lines:	1598
Uncovered lines:	1760
Coverable lines:	3358
Total lines:	6092
Line coverage:	47.5% (1598 of 3358)
Covered branches:	383
Total branches:	995
Branch coverage:	38.4% (383 of 995)
Covered methods:	220
Total methods:	531
Method coverage:	41.4% (220 of 531)

Risk Hotspots

Assembly	Class	Method	Cyclomatic complexity
DIKUArcade	DIKUArcade.Input.Languages.DanishKeyTransformer	TransformKey(...)	194
DIKUArcade	DIKUArcade.Physics.CollisionDetection	Aabb(...)	48
Breakout	Breakout.Players.Player	ProcessEvent(...)	46

Coverage

Name	Covered	Uncovered	Coverable	Total	Line coverage	Branch coverage	Method coverage
Breakout	**1152**	**117**	**1269**	**2176**	**90.7%**	**95.3%**	**87.1%**
Breakout.Balls.Ball	9	3	12	32	75%		75%
Breakout.Balls.BallCreator	7	0	7	20	100%		100%
Breakout.Blocks.Block	19	3	22	46	86.3%		80%
Breakout.Blocks.BlockCreator	7	0	7	25	100%	100%	100%

Breakout.Blocks.DefaultBlock	9	0	9	21	100%	100%	100%
Breakout.Blocks.Hardened	15	1	16	31	93.7%	83.3%	100%
Breakout.Blocks.PowerupBlock	14	0	14	26	100%	100%	100%
Breakout.Blocks.Unbreakable	5	0	5	16	100%		100%
Breakout.BreakoutBus	14	0	14	27	100%	100%	100%
Breakout.Collisions.BlockCollision	38	0	38	53	100%	100%	100%
Breakout.Collisions.PlayerCollision	33	0	33	56	100%	100%	100%
Breakout.Collisions.PowerUpCollision	13	0	13	27	100%	100%	100%
Breakout.Collisions.WallCollision	38	0	38	53	100%	100%	100%
Breakout.Constants	2	0	2	11	100%		100%
Breakout.Game	0	24	24	49	0%	0%	0%
Breakout.Health	31	3	34	59	91.1%	100%	80%
Breakout.Levels.LevelCreator	53	0	53	90	100%	91.6%	100%
Breakout.Levels.LevelManager	87	12	99	159	87.8%	86.6%	93.7%
Breakout.Levels.LevelReader	76	0	76	111	100%	93.7%	100%
Breakout.Players.Player	70	6	76	112	92.1%	100%	80%
Breakout.Points	31	3	34	66	91.1%	100%	85.7%
Breakout.Powerups.HardBall	15	0	15	27	100%		100%
Breakout.Powerups.LifeLoss	9	0	9	19	100%		100%
Breakout.Powerups.LifePlus	10	0	10	21	100%		100%
Breakout.Powerups.PlayerSpeed	15	0	15	27	100%		100%
Breakout.Powerups.Powerup	8	0	8	23	100%	100%	100%
Breakout.Powerups.PowerUpCreator	34	0	34	56	100%	100%	100%
Breakout.Powerups.SlimJim	15	0	15	27	100%		100%
Breakout.Powerups.Split	9	0	9	19	100%		100%
Breakout.Powerups.Wide	15	0	15	27	100%		100%
Breakout.Program	0	5	5	9	0%		0%
Breakout.States.GameLost	74	15	89	139	83.1%	91.6%	77.7%
Breakout.States.GamePaused	73	8	81	129	90.1%	100%	81.8%
Breakout.States.GameRunning	98	6	104	167	94.2%	100%	93.3%
Breakout.States.GameWon	75	15	90	138	83.3%	91.6%	77.7%
Breakout.States.MainMenu	57	13	70	114	81.4%	91.6%	80%
Breakout.States.StateMachine	29	0	29	54	100%	100%	100%
Breakout.States.StateTransformer	9	0	9	27	100%	100%	100%
Breakout.Timer	36	0	36	63	100%	100%	100%
DIKUArcade	**446**	**1643**	**2089**	**3916**	**21.3%**	**10.8%**	**19.7%**
DIKUArcade.DIKUGame	0	32	32	73	0%	0%	0%
DIKUArcade.Entities.DynamicShape	21	7	28	51	75%		71.4%

DIKUArcade.Entities.Entity	13	6	19	40	68.4%		71.4%
DIKUArcade.Entities.EntityContainer	0	63	63	132	0%	0%	0%
DIKUArcade.Entities.EntityContainer<T>	39	12	51	113	76.4%	75%	73.3%
DIKUArcade.Entities.Shape	6	55	61	114	9.8%	25%	15%
DIKUArcade.Entities.StationaryShape	4	7	11	23	36.3%		33.3%
DIKUArcade.Events.GameEventBus	86	88	174	285	49.4%	31.6%	57.8%
DIKUArcade.Events.GameEventQueue<T>	17	13	30	97	56.6%	100%	41.6%
DIKUArcade.Events.Generic.GameEventBus<T>	0	174	174	287	0%	0%	0%
DIKUArcade.Events.Generic.TimedGameEvent<T>	0	13	13	46	0%		0%
DIKUArcade.Events.TimedGameEvent	9	4	13	43	69.2%		75%
DIKUArcade.Graphics.Animation	0	15	15	44	0%		0%
DIKUArcade.Graphics.AnimationContainer	0	40	40	72	0%	0%	0%
DIKUArcade.Graphics.Camera	0	12	12	26	0%		0%
DIKUArcade.Graphics.ChaseCamera	0	19	19	50	0%	0%	0%
DIKUArcade.Graphics.DynamicCamera	0	46	46	79	0%	0%	0%
DIKUArcade.Graphics.FollowCamera	0	13	13	23	0%	0%	0%
DIKUArcade.Graphics.Image	3	12	15	28	20%		20%
DIKUArcade.Graphics.ImageStride	0	87	87	191	0%	0%	0%
DIKUArcade.Graphics.NoImage	0	3	3	11	0%		0%
DIKUArcade.Graphics.StaticCamera	0	4	4	14	0%		0%
DIKUArcade.Graphics.Text	79	62	141	300	56%	7.6%	47%
DIKUArcade.Graphics.Texture	38	92	130	226	29.2%	21.4%	37.5%
DIKUArcade.GUI.Window	6	172	178	377	3.3%	0%	3.8%
DIKUArcade.GUI.WindowArgs	0	7	7	27	0%		0%
DIKUArcade.Input.Languages.DanishKeyTransformer	0	125	125	155	0%	0%	0%
DIKUArcade.Math.Vec2D	0	39	39	61	0%		0%
DIKUArcade.Math.Vec2F	8	37	45	70	17.7%		21.4%
DIKUArcade.Math.Vec2I	0	36	36	57	0%		0%
DIKUArcade.Math.Vec3D	0	38	38	60	0%		0%
DIKUArcade.Math.Vec3F	0	38	38	60	0%		0%
DIKUArcade.Math.Vec3I	5	33	38	60	13.1%		9%
DIKUArcade.Math.Vec4D	0	40	40	63	0%		0%
DIKUArcade.Math.Vec4F	0	40	40	63	0%		0%
DIKUArcade.Math.Vec4I	0	40	40	63	0%		0%
DIKUArcade.Physics.CollisionData	3	0	3	23	100%		100%
DIKUArcade.Physics.CollisionDetection	84	27	111	184	75.6%	68.7%	50%
DIKUArcade.Timers.GameTimer	0	57	57	100	0%	0%	0%
DIKUArcade.Timers.StaticTimer	8	21	29	59	27.5%	0%	28.5%

DIKUArcade.Timers.TimePeriod	8	8	16	30	50%	16.6%	60%
DIKUArcade.Utilities.FileIO	9	0	9	23	100%	100%	100%
DIKUArcade.Utilities.RandomGenerator	0	6	6	13	0%	0%	0%

Breakout.Balls.Ball

Summary

Class:	Breakout.Balls.Ball
Assembly:	Breakout
File(s):	/home/student/SU23Guest/DIKUGames/Breakout/Ball/Ball.cs
Covered lines:	9
Uncovered lines:	3
Coverable lines:	12
Total lines:	32
Line coverage:	75% (9 of 12)
Covered branches:	0
Total branches:	0
Covered methods:	3
Total methods:	4
Method coverage:	75% (3 of 4)

Metrics

Method	Branch coverage	Cyclomatic complexity	Line coverage
get_Shape()	100%	1	100%
.ctor(...)	100%	1	100%
Move()	100%	1	100%
Render()	100%	1	0%

File(s)

/home/student/SU23Guest/DIKUGames/Breakout/Ball/Ball.cs

#	Line	Line coverage
	1	`using DIKUArcade.Entities;`
	2	`using DIKUArcade.Graphics;`
	3	`namespace Breakout.Balls;`
	4	`/// <summary>`
	5	`/// A ball entity that can move.`
	6	`/// Static collision classes can be used to detect collisions with the ball and objects`
	7	`/// </summary>`

```
      8    public class Ball : Entity {
      9        private DynamicShape shape;
     10
     11        public DynamicShape _Shape {
1122 12            get {
1122 13                return shape;
1122 14            }
     15        }
 592 16        public Ball(DynamicShape shape, IBaseImage image) : base(shape, image) {
 296 17            this.shape = shape;
 296 18        }
     19        /// <summary>
     20        /// Moves the ball along it's directional vector contained in shape.
     21        /// </summary>
  78 22        public void Move() {
  78 23            shape.Move();
  78 24        }
     25
   0 26        public void Render() {
   0 27            this.RenderEntity();
   0 28        }
     29    }
     30
     31
     32
```

Breakout.Balls.BallCreator

Summary

Class:	Breakout.Balls.BallCreator
Assembly:	Breakout
File(s):	/home/student/SU23Guest/DIKUGames/Breakout/Ball/BallCreator.cs
Covered lines:	7
Uncovered lines:	0
Coverable lines:	7
Total lines:	20
Line coverage:	100% (7 of 7)
Covered branches:	0
Total branches:	0
Covered methods:	1
Total methods:	1
Method coverage:	100% (1 of 1)

Metrics

Method	Branch coverage	Cyclomatic complexity	Line coverage
CreateBall(...)	100%	1	100%

File(s)

/home/student/SU23Guest/DIKUGames/Breakout/Ball/BallCreator.cs

#	Line	Line coverage
	1	`using DIKUArcade.Entities;`
	2	`using DIKUArcade.Graphics;`
	3	`using DIKUArcade.Math;`
	4	`using System.IO;`
	5	`namespace Breakout.Balls;`
	6	`/// <summary>`
	7	`///  Creates a ball entity`
	8	`/// </summary>`
	9	`public static class BallCreator {`
	10	`    /// <summary>`

```
        11        ///  Creates a ball using a position and a directional vector
        12        /// </summary>
277     13        public static Ball CreateBall(Vec2F pos, Vec2F dir) {
277     14            return new Ball(new DynamicShape(
277     15                pos,
277     16                new Vec2F(0.03f, 0.03f),
277     17                dir),
277     18            new Image(Path.Combine("..", "Breakout", "Assets", "Images", "ball2.png")));
277     19        }
        20    }
```

Breakout.Blocks.Block

Summary

Class:	Breakout.Blocks.Block
Assembly:	Breakout
File(s):	/home/student/SU23Guest/DIKUGames/Breakout/Blocks/Block.cs
Covered lines:	19
Uncovered lines:	3
Coverable lines:	22
Total lines:	46
Line coverage:	86.3% (19 of 22)
Covered branches:	0
Total branches:	0
Covered methods:	4
Total methods:	5
Method coverage:	80% (4 of 5)

Metrics

Method	Branch coverage	Cyclomatic complexity	Line coverage
.ctor(...)	100%	1	100%
get_Value()	100%	1	100%
get_Health()	100%	1	100%
GivePoints()	100%	1	100%
Render()	100%	1	0%

File(s)

/home/student/SU23Guest/DIKUGames/Breakout/Blocks/Block.cs

#	Line	Line coverage
	1	`using DIKUArcade.Entities;`
	2	`using DIKUArcade.Events;`
	3	`using DIKUArcade.Graphics;`
	4	`using DIKUArcade.Math;`
	5	`using System.IO;`
	6	`namespace Breakout.Blocks;`

```
        7    /// <summary>
        8    /// An abstract block type
        9    /// </summary>
       10    public abstract class Block : Entity {
 4412  11        protected int value = 10;
       12        /// <summary>
       13        /// Amount of points given to player when block is destroyed.
       14        /// </summary>
       15        public int Value {
    4  16            get {
    4  17                return value;
    4  18            }
       19        }
 4412  20        protected int health = 1;
       21        public int Health {
   34  22            get {
   34  23                return health;
   34  24            }
       25        }
       26        protected Vec2F position;
       27        public Block(Shape shape, string imageFile) :
 4412  28        base(shape, new Image(
 8824  29            Path.Combine("..", "Breakout", "Assets", "Images", imageFile))) {
 4412  30            position = new Vec2F(shape.Position.X, shape.Position.Y);
 4412  31        }
       32        /// <summary>
       33        /// Decreases Block health, if health is less than 1 the block is marked for deletion.
       34        /// </summary>
       35        public abstract void LoseHealth(int amount);
   12  36        protected void GivePoints() {
   12  37            BreakoutBus.GetBus().RegisterEvent(new GameEvent {
   12  38                EventType = GameEventType.StatusEvent,
   12  39                Message = "GET POINTS",
   12  40                IntArg1 = value
   12  41            });
   12  42        }
    0  43        public void Render() {
    0  44            RenderEntity();
    0  45        }
       46    }
```

Breakout.Blocks.BlockCreator

Summary

Class:	Breakout.Blocks.BlockCreator
Assembly:	Breakout
File(s):	/home/student/SU23Guest/DIKUGames/Breakout/Blocks/BlockCreator.cs
Covered lines:	7
Uncovered lines:	0
Coverable lines:	7
Total lines:	25
Line coverage:	100% (7 of 7)
Covered branches:	6
Total branches:	6
Branch coverage:	100% (6 of 6)
Covered methods:	1
Total methods:	1
Method coverage:	100% (1 of 1)

Metrics

Method	Branch coverage	Cyclomatic complexity	Line coverage
CreateBlock(...)	100%	6	100%

File(s)

/home/student/SU23Guest/DIKUGames/Breakout/Blocks/BlockCreator.cs

```
# Line    Line coverage
  1       using DIKUArcade.Entities;
  2       namespace Breakout.Blocks;
  3       /// <summary>
  4       /// A class for creating block types depending on meta data.
  5       /// </summary>
  6       public static class BlockCreator {
  7           /// <summary>
  8           /// Creates a block
  9           /// </summary>
```

```
        /// <param name="shape">The shape of the block.</param>
        /// <param name="image">Image file used for the block.</param>
        /// <param name="meta">Metadata that determines what type of block should be created.</param>
        public static Block CreateBlock(Shape shape, string image, string meta) {
            switch (meta) {
                case "Hardened":
                    return new Hardened(shape, image);
                case "Unbreakable":
                    return new Unbreakable(shape, image);
                case "PowerUp":
                    return new PowerupBlock(shape, image);
                default:
                    return new DefaultBlock(shape, image);
            }
        }
    }
```

Breakout.Blocks.DefaultBlock

Summary

Class:	Breakout.Blocks.DefaultBlock
Assembly:	Breakout
File(s):	/home/student/SU23Guest/DIKUGames/Breakout/Blocks/DefaultBlock.cs
Covered lines:	9
Uncovered lines:	0
Coverable lines:	9
Total lines:	21
Line coverage:	100% (9 of 9)
Covered branches:	2
Total branches:	2
Branch coverage:	100% (2 of 2)
Covered methods:	2
Total methods:	2
Method coverage:	100% (2 of 2)

Metrics

Method	Branch coverage	Cyclomatic complexity	Line coverage
.ctor(...)	100%	1	100%
LoseHealth(...)	100%	2	100%

File(s)

/home/student/SU23Guest/DIKUGames/Breakout/Blocks/DefaultBlock.cs

```
#  Line   Line coverage
     1    using DIKUArcade.Entities;
     2    using DIKUArcade.Events;
     3
     4    namespace Breakout.Blocks;
     5    /// <summary>
     6    /// Default block has 1 health points and grants player 10 points when destroyed.
     7    /// </summary>
     8    public class DefaultBlock : Block {
```

```
8628        public DefaultBlock(Shape shape, string imageFile) : base(shape, imageFile) {
4314        }
            /// <summary>
            /// Decreases health, if health is 0 the block is marked for deletion.
            /// </summary>
   8        public override void LoseHealth(int amount) {
   8            health -= amount;
  14            if (health == 0) {
   6                DeleteEntity();
   6                GivePoints();
   6            }
   8        }
            }
```

Breakout.Blocks.Hardened

Summary

Class:	Breakout.Blocks.Hardened
Assembly:	Breakout
File(s):	/home/student/SU23Guest/DIKUGames/Breakout/Blocks/Hardened.cs
Covered lines:	15
Uncovered lines:	1
Coverable lines:	16
Total lines:	31
Line coverage:	93.7% (15 of 16)
Covered branches:	5
Total branches:	6
Branch coverage:	83.3% (5 of 6)
Covered methods:	2
Total methods:	2
Method coverage:	100% (2 of 2)

Metrics

Method	Branch coverage	Cyclomatic complexity	Line coverage
.ctor(...)	100%	1	100%
LoseHealth(...)	83.33%	6	88.88%

File(s)

/home/student/SU23Guest/DIKUGames/Breakout/Blocks/Hardened.cs

#	Line	Line coverage
	1	`using DIKUArcade.Entities;`
	2	`using DIKUArcade.Graphics;`
	3	`using DIKUArcade.Math;`
	4	`using System.IO;`
	5	`namespace Breakout.Blocks;`
	6	`/// <summary>`
	7	`/// Block has 2 health points and grants player 20 points when destroyed.`
	8	`/// </summary>`

```
    9  public class Hardened : Block {
   10      private string DamagedImg;
   11      private string Damaged;
40 12      public Hardened(Shape shape, string imageFile) : base(shape, imageFile) {
20 13          position = new Vec2F(shape.Position.X, shape.Position.Y);
20 14          DamagedImg = imageFile.Insert(imageFile.Length - 4, "-damaged");
20 15          Damaged = Path.Combine("..", "Breakout", "Assets", "Images", DamagedImg);
20 16          health = 2;
20 17          value = 20;
20 18      }
   19      /// <summary>
   20      /// Decreases Block health, if health is 0 the block is marked for deletion
   21      /// </summary>
 6 22      public override void LoseHealth(int amount) {
 6 23          health -= amount;
 6 24          if (health == 1 && File.Exists(Damaged)) {
 0 25              Image = new Image(Damaged);
 9 26          } else if (health == 0) {
 3 27              DeleteEntity();
 3 28              GivePoints();
 3 29          }
 6 30      }
   31  }
```

Breakout.Blocks.PowerupBlock

Summary

Class:	Breakout.Blocks.PowerupBlock
Assembly:	Breakout
File(s):	/home/student/SU23Guest/DIKUGames/Breakout/Blocks/PowerUpBlock.cs
Covered lines:	14
Uncovered lines:	0
Coverable lines:	14
Total lines:	26
Line coverage:	100% (14 of 14)
Covered branches:	2
Total branches:	2
Branch coverage:	100% (2 of 2)
Covered methods:	2
Total methods:	2
Method coverage:	100% (2 of 2)

Metrics

Method	Branch coverage	Cyclomatic complexity	Line coverage
.ctor(...)	100%	1	100%
LoseHealth(...)	100%	2	100%

File(s)

/home/student/SU23Guest/DIKUGames/Breakout/Blocks/PowerUpBlock.cs

```
   #  Line   Line coverage
      1     using DIKUArcade.Entities;
      2     using DIKUArcade.Events;
      3     namespace Breakout.Blocks;
      4     /// <summary>
      5     /// Has 1 health points and grants player 10 points when destroyed and drops a powerup/hazard.
      6     /// </summary>
      7     public class PowerupBlock : Block {
 82   8         public PowerupBlock(Shape shape, string imageFile) : base(shape, imageFile) {
```

```
41  9        }
    10       /// <summary>
    11       /// Decreases Block health, if health is 0 the block is marked for deletion
    12       /// and a power-up will be created
    13       /// </summary>
 5  14       public override void LoseHealth(int amount) {
 5  15           health -= amount;
 8  16           if (health == 0) {
 3  17               BreakoutBus.GetBus().RegisterEvent(new GameEvent {
 3  18                   EventType = GameEventType.StatusEvent,
 3  19                   Message = "SPAWN POWERUP",
 3  20                   ObjectArg1 = (object) Shape.Position
 3  21               });
 3  22               DeleteEntity();
 3  23               GivePoints();
 3  24           }
 5  25       }
    26   }
```

Breakout.Blocks.Unbreakable

Summary

Class:	Breakout.Blocks.Unbreakable
Assembly:	Breakout
File(s):	/home/student/SU23Guest/DIKUGames/Breakout/Blocks/Unbreakable.cs
Covered lines:	5
Uncovered lines:	0
Coverable lines:	5
Total lines:	16
Line coverage:	100% (5 of 5)
Covered branches:	0
Total branches:	0
Covered methods:	2
Total methods:	2
Method coverage:	100% (2 of 2)

Metrics

Method	Branch coverage	Cyclomatic complexity	Line coverage
.ctor(...)	100%	1	100%
LoseHealth(...)	100%	1	100%

File(s)

/home/student/SU23Guest/DIKUGames/Breakout/Blocks/Unbreakable.cs

#	Line	Line coverage
	1	`using DIKUArcade.Entities;`
	2	`using DIKUArcade.Math;`
	3	`namespace Breakout.Blocks;`
	4	`/// <summary>`
	5	`/// Unbreakable block, can't be destroyed`
	6	`/// </summary>`
	7	`public class Unbreakable : Block {`
74	8	`    public Unbreakable(Shape shape, string imageFile) : base(shape, imageFile) {`
37	9	`        position = new Vec2F(shape.Position.X, shape.Position.Y);`

```
37  10        }
    11        /// <summary>
    12        /// Does nothing, beacuse unbreakable blocks can't lose health and die
    13        /// </summary>
20  14        public override void LoseHealth(int amount) {
20  15        }
    16    }
```

Breakout.BreakoutBus

Summary

Class:	Breakout.BreakoutBus
Assembly:	Breakout
File(s):	/home/student/SU23Guest/DIKUGames/Breakout/BreakoutBus.cs
Covered lines:	14
Uncovered lines:	0
Coverable lines:	14
Total lines:	27
Line coverage:	100% (14 of 14)
Covered branches:	2
Total branches:	2
Branch coverage:	100% (2 of 2)
Covered methods:	1
Total methods:	1
Method coverage:	100% (1 of 1)

Metrics

Method	Branch coverage	Cyclomatic complexity	Line coverage
GetBus()	100%	2	100%

File(s)

/home/student/SU23Guest/DIKUGames/Breakout/BreakoutBus.cs

#	Line	Line coverage
	1	`using DIKUArcade.Events;`
	2	`using System.Collections.Generic;`
	3	`namespace Breakout;`
	4	`/// <summary>`
	5	`/// A static eventBus used for registering and handling gameEvents`
	6	`/// </summary>`
	7	`public static class BreakoutBus {`
	8	`    private static GameEventBus eventBus;`
	9	`    /// <summary>`

```
        /// Retrieves the static Eventbus, allowing for use of eventBus methods.
        /// </summary>
476     public static GameEventBus GetBus() {
477         if (BreakoutBus.eventBus == null) {
  1             BreakoutBus.eventBus = new GameEventBus();
  1             BreakoutBus.GetBus().InitializeEventBus(
  1                 new List<GameEventType> {
  1                     GameEventType.InputEvent,
  1                     GameEventType.WindowEvent,
  1                     GameEventType.PlayerEvent,
  1                     GameEventType.GameStateEvent,
  1                     GameEventType.StatusEvent
  1                 });
  1         }
476         return BreakoutBus.eventBus;
476     }
    }
```

Breakout.Collisions.BlockCollision

Summary

Class:	Breakout.Collisions.BlockCollision
Assembly:	Breakout
File(s):	/home/student/SU23Guest/DIKUGames/Breakout/Collisions/BlockCollision.cs
Covered lines:	38
Uncovered lines:	0
Coverable lines:	38
Total lines:	53
Line coverage:	100% (38 of 38)
Covered branches:	12
Total branches:	12
Branch coverage:	100% (12 of 12)
Covered methods:	1
Total methods:	1
Method coverage:	100% (1 of 1)

Metrics

Method	Branch coverage	Cyclomatic complexity	Line coverage
Collide(...)	100%	12	100%

File(s)

/home/student/SU23Guest/DIKUGames/Breakout/Collisions/BlockCollision.cs

#	Line	Line coverage
	1	`using Breakout.Balls;`
	2	`using Breakout.Blocks;`
	3	`using DIKUArcade.Entities;`
	4	`using DIKUArcade.Math;`
	5	`using DIKUArcade.Physics;`
	6	`namespace Breakout.Collisions;`
	7	`/// <summary>`
	8	`/// Handles collisions between balls and blocks`
	9	`/// </summary>`

```
        10    public static class BlockCollision {
        11        /// <summary>
        12        /// Will check for collsions between ball and blocks,
        13        /// blocks lose health and ball direction is changed when collisions occur.
        14        /// </summary>
 21     15        public static void Collide(EntityContainer<Ball> balls, EntityContainer<Block> blocks, bool hardBalls) {
 42     16            balls.Iterate(ball => {
 21     17                // Iterating through every block deletes blocks marked for deletion
857     18                blocks.Iterate(block => {
836     19                    CollisionData blockCollision = CollisionDetection.Aabb(ball._Shape, block.Shape);
839     20                    if (blockCollision.Collision) { // True if there is collision between the ball and block
  4     21                        if (hardBalls) {
 21     22                            // If hardball powerup is active block should be deleted
  1     23                            block.LoseHealth(block.Health);
  1     24                            block.DeleteEntity();
  3     25                        } else {
 21     26                            // If hardball isnt active the block should lose 1 health
  2     27                            block.LoseHealth(1);
  2     28                        }
  3     29                        CollisionDirection collisionDirection = blockCollision.CollisionDir;
  3     30                        Vec2F currentDirection = ball._Shape.Direction;
  3     31                        switch (collisionDirection) {
 21     32                            case CollisionDirection.CollisionDirUp:
 21     33                            case CollisionDirection.CollisionDirDown:
  2     34                                if (!hardBalls) {
 21     35                                    // If hardball is active, ball shouldnt change direction.
  1     36                                    ball._Shape.ChangeDirection(
  1     37                                        new Vec2F(currentDirection.X, -currentDirection.Y));
  1     38                                }
  1     39                                break;
 21     40                            case CollisionDirection.CollisionDirLeft:
 21     41                            case CollisionDirection.CollisionDirRight:
  3     42                                if (!hardBalls) {
 21     43                                    // If hardball is active, ball shouldnt change direction.
  1     44                                    ball._Shape.ChangeDirection(
  1     45                                        new Vec2F(-currentDirection.X, currentDirection.Y));
  1     46                                }
  2     47                                break;
 21     48                        }
  3     49                    }
```

```
857    50                });
 42    51            });
 21    52        }
       53    }
```

Breakout.Collisions.PlayerCollision

Summary

Class:	Breakout.Collisions.PlayerCollision
Assembly:	Breakout
File(s):	/home/student/SU23Guest/DIKUGames/Breakout/Collisions/PlayerCollision.cs
Covered lines:	33
Uncovered lines:	0
Coverable lines:	33
Total lines:	56
Line coverage:	100% (33 of 33)
Covered branches:	16
Total branches:	16
Branch coverage:	100% (16 of 16)
Covered methods:	1
Total methods:	1
Method coverage:	100% (1 of 1)

Metrics

Method	Branch coverage	Cyclomatic complexity	Line coverage
Collide(...)	100%	16	100%

File(s)

/home/student/SU23Guest/DIKUGames/Breakout/Collisions/PlayerCollision.cs

#	Line	Line coverage
	1	`using Breakout.Balls;`
	2	`using Breakout.Players;`
	3	`using DIKUArcade.Entities;`
	4	`using DIKUArcade.Math;`
	5	`using DIKUArcade.Physics;`
	6	`namespace Breakout.Collisions;`
	7	`/// <summary>`
	8	`/// Handles collisions between balls and the player`
	9	`/// </summary>`

```
        10    public static class PlayerCollision {
        11        /// <summary>
        12        /// Will check for collsions between ball and the player,
        13        /// depending on where the ball hit the player, the balls directional vector is changed.
        14        /// </summary>
 45     15        public static bool Collide(EntityContainer<Ball> balls, Player player) {
 45     16            bool hit = false;
 45     17            Vec2F vec = new Vec2F(0.0f, 0.015f);
 45     18            Vec2F vec20 = new Vec2F(-0.0051f, 0.01409f);
 45     19            Vec2F revVec20 = new Vec2F(0.0051f, 0.01409f);
 45     20            Vec2F vec45 = new Vec2F(-0.0106f, 0.0106f);
 45     21            Vec2F revVec45 = new Vec2F(0.0106f, 0.0106f);
 45     22            float playerposx = player.Shape.Position.X;
140     23            balls.Iterate(ball => {
 45     24                // Iterating through every block
 95     25                CollisionData collision = CollisionDetection.Aabb(ball._Shape, player.Shape);
100     26                if (collision.Collision) { // True if there is collision between the ball and player
  5     27                    hit = true;
  5     28                    float ballx = ball._Shape.Position.X + (ball._Shape.Extent.X / 2); //Middle of ball
  5     29                    float playerExtentX = player.Shape.Extent.X;
  6     30                    if (ballx < playerposx + (playerExtentX / 5)) {
  1     31                        ball._Shape.ChangeDirection(vec45);
  5     32                    } else if (ballx < playerposx + (playerExtentX / 5) * 2 &&
  5     33                                ballx > playerposx + (player.Shape.Extent.X / 5)) {
  1     34                        ball._Shape.ChangeDirection(vec20);
  4     35                    } else if (ballx < playerposx + (playerExtentX / 5) * 3 &&
  4     36                                ballx > playerposx + (playerExtentX / 5) * 2) {
  1     37                        ball._Shape.ChangeDirection(vec);
  3     38                    } else if (ballx < playerposx + (playerExtentX / 5) * 4 &&
  3     39                                ballx > playerposx + (playerExtentX / 5) * 3) {
  1     40                        ball._Shape.ChangeDirection(revVec20);
  2     41                    } else {
  1     42                        ball._Shape.ChangeDirection(revVec45);
  1     43                    }
  5     44                }
140     45            });
 45     46            return hit;
 45     47        }
        48    }
        49
```

```
50
51
52
53
54
55
56
```

Breakout.Collisions.PowerUpCollision

Summary

Class:	Breakout.Collisions.PowerUpCollision
Assembly:	Breakout
File(s):	/home/student/SU23Guest/DIKUGames/Breakout/Collisions/PowerUpCollision.cs
Covered lines:	13
Uncovered lines:	0
Coverable lines:	13
Total lines:	27
Line coverage:	100% (13 of 13)
Covered branches:	2
Total branches:	2
Branch coverage:	100% (2 of 2)
Covered methods:	1
Total methods:	1
Method coverage:	100% (1 of 1)

Metrics

Method	Branch coverage	Cyclomatic complexity	Line coverage
Collide(...)	100%	2	100%

File(s)

/home/student/SU23Guest/DIKUGames/Breakout/Collisions/PowerUpCollision.cs

#	Line	Line coverage
	1	`using Breakout.Players;`
	2	`using DIKUArcade.Entities;`
	3	`using DIKUArcade.Physics;`
	4	`using Breakout.Powerups;`
	5	`namespace Breakout.Collisions;`
	6	`/// <summary>`
	7	`/// Handles collisions between the player and powerups`
	8	`/// </summary>`
	9	`public static class PowerUpCollision {`

```
        10        /// <summary>
        11        /// Will check for collsions between powerup and the player,
        12        /// depending on where the ball hit the player, the balls directional vector is changed.
        13        /// </summary>
22      14        public static bool Collide(EntityContainer<Powerup> powerups, Player player) {
22      15            bool hit = false;
33      16            powerups.Iterate(powerup => {
11      17                CollisionData collision = CollisionDetection.Aabb(
11      18                    (DynamicShape) powerup.Shape, player.Shape);
21      19                if (collision.Collision) {
10      20                    powerup.Effect();
10      21                    hit = true;
10      22                    powerup.DeleteEntity();
10      23                }
33      24            });
22      25            return hit;
22      26        }
        27    }
```

Breakout.Collisions.WallCollision

Summary

Class:	Breakout.Collisions.WallCollision
Assembly:	Breakout
File(s):	/home/student/SU23Guest/DIKUGames/Breakout/Collisions/WallCollision.cs
Covered lines:	38
Uncovered lines:	0
Coverable lines:	38
Total lines:	53
Line coverage:	100% (38 of 38)
Covered branches:	10
Total branches:	10
Branch coverage:	100% (10 of 10)
Covered methods:	5
Total methods:	5
Method coverage:	100% (5 of 5)

Metrics

Method	Branch coverage	Cyclomatic complexity	Line coverage
Collide(...)	100%	2	100%
CollideLeftWall(...)	100%	2	100%
CollideRightWall(...	100%	2	100%
CollideTopWall(...)	100%	2	100%
CollideBottom(...)	100%	2	100%

File(s)

/home/student/SU23Guest/DIKUGames/Breakout/Collisions/WallCollision.cs

#	Line	Line coverage
	1	`using Breakout.Balls;`
	2	`using DIKUArcade.Entities;`
	3	`using DIKUArcade.Events;`
	4	`namespace Breakout.Collisions;`
	5	`/// <summary>`

```
        6    /// Handles collisions between balls and walls
        7    /// </summary>
        8    public static class WallCollision {
        9        /// <summary>
       10        /// Will check for collsions between ball and walls.
       11        /// If ball hits a wall it's send the opposite direction.
       12        /// If balls hits bottom, lose health event and a new ball event is send.
       13        /// </summary>
20     14        public static void Collide(EntityContainer<Ball> balls) {
39     15            balls.Iterate(ball => {
19     16                CollideLeftWall(ball);
19     17                CollideRightWall(ball);
19     18                CollideTopWall(ball);
19     19                CollideBottom(ball);
39     20            });
25     21            if (balls.CountEntities() == 0) {
 5     22                BreakoutBus.GetBus().RegisterEvent(new GameEvent {
 5     23                    EventType = GameEventType.StatusEvent,
 5     24                    Message = "LOSE HEALTH"
 5     25                });
 5     26                BreakoutBus.GetBus().RegisterEvent(new GameEvent {
 5     27                    EventType = GameEventType.StatusEvent,
 5     28                    Message = "NEW BALL"
 5     29                });
 5     30            }
       31
20     32        }
19     33        private static void CollideLeftWall(Ball ball) {
20     34            if (ball._Shape.Position.X < 0) {
 1     35                ball._Shape.Direction.X = -ball._Shape.Direction.X;
 1     36            }
19     37        }
19     38        private static void CollideRightWall(Ball ball) {
20     39            if (ball._Shape.Position.X + ball._Shape.Extent.X > 1) {
 1     40                ball._Shape.Direction.X = -ball._Shape.Direction.X;
 1     41            }
19     42        }
19     43        private static void CollideTopWall(Ball ball) {
20     44            if (ball._Shape.Position.Y + ball._Shape.Extent.Y > 1) {
 1     45                ball._Shape.Direction.Y = -ball._Shape.Direction.Y;
```

```
 1    46            }
19    47        }
19    48        private static void CollideBottom(Ball ball) {
21    49            if (ball._Shape.Position.Y <= 0.0 - ball._Shape.Extent.Y) {
 2    50                ball.DeleteEntity();
 2    51            }
19    52        }
      53    }
```

Breakout.Constants

Summary

Class:	Breakout.Constants
Assembly:	Breakout
File(s):	/home/student/SU23Guest/DIKUGames/Breakout/Constants.cs
Covered lines:	2
Uncovered lines:	0
Coverable lines:	2
Total lines:	11
Line coverage:	100% (2 of 2)
Covered branches:	0
Total branches:	0
Covered methods:	1
Total methods:	1
Method coverage:	100% (1 of 1)

Metrics

Method	Branch coverage	Cyclomatic complexity	Line coverage
.cctor()	100%	1	100%

File(s)

/home/student/SU23Guest/DIKUGames/Breakout/Constants.cs

#	Line	Line coverage
	1	`using DIKUArcade.Utilities;`
	2	`using System.IO;`
	3	`namespace Breakout;`
	4	`/// <summary> Class for conatining constants </summary>`
	5	`/// <remarks> Used for indicating path to assets </remarks>`
	6	`public static class Constants {`
	7	`    /// <summary> This is to get the path of the Breakout/Assets </summary>`
	8	`    /// <remarks> We need this for testing, to make sure we are using the same assets </remarks>`
1	9	`    public static readonly string MAIN_PATH =`
1	10	`        Path.Combine(Directory.GetParent(FileIO.GetProjectPath())!.FullName, "Breakout");`

```
}
```

Breakout.Game

Summary

Class:	Breakout.Game
Assembly:	Breakout
File(s):	/home/student/SU23Guest/DIKUGames/Breakout/Game.cs
Covered lines:	0
Uncovered lines:	24
Coverable lines:	24
Total lines:	49
Line coverage:	0% (0 of 24)
Covered branches:	0
Total branches:	4
Branch coverage:	0% (0 of 4)
Covered methods:	0
Total methods:	5
Method coverage:	0% (0 of 5)

Metrics

Method	Branch coverage	Cyclomatic complexity	Line coverage
.ctor(...)	100%	1	0%
ProcessEvent(...)	0%	4	0%
KeyHandler(...)	100%	1	0%
Render()	100%	1	0%
Update()	100%	1	0%

File(s)

/home/student/SU23Guest/DIKUGames/Breakout/Game.cs

#	Line	Line coverage
	1	`using Breakout.States;`
	2	`using DIKUArcade;`
	3	`using DIKUArcade.Events;`
	4	`using DIKUArcade.GUI;`
	5	`using DIKUArcade.Input;`

```
      6
      7  namespace Breakout;
      8  /// <summary>
      9  ///  Class responsible for handling a statemachine and keyinputs
     10  /// </summary>
     11  public class Game : DIKUGame, IGameEventProcessor {
     12      private StateMachine stateMachine;
0    13      public Game(WindowArgs windowArgs) : base(windowArgs) {
0    14          window.SetKeyEventHandler(KeyHandler);
0    15          BreakoutBus.GetBus().Subscribe(GameEventType.InputEvent, this);
0    16          BreakoutBus.GetBus().Subscribe(GameEventType.WindowEvent, this);
     17
0    18          stateMachine = new StateMachine();
0    19          BreakoutBus.GetBus().Subscribe(GameEventType.GameStateEvent, stateMachine);
0    20      }
     21      /// <summary>
     22      ///  processes windowevents, can close window
     23      /// </summary>
0    24      public void ProcessEvent(GameEvent gameEvent) {
0    25          if (gameEvent.EventType == GameEventType.WindowEvent) {
0    26              switch (gameEvent.StringArg1) {
     27                  case "WINDOW CLOSE":
0    28                      window.CloseWindow();
0    29                      break;
     30              }
0    31          }
0    32      }
0    33      private void KeyHandler(KeyboardAction action, KeyboardKey key) {
0    34          stateMachine.ActiveState.HandleKeyEvent(action, key);
0    35      }
     36      /// <summary>
     37      ///  Renders the active state
     38      /// </summary>
0    39      public override void Render() {
0    40          stateMachine.ActiveState.RenderState();
0    41      }
     42      /// <summary>
     43      ///  Updates the active state
     44      /// </summary>
0    45      public override void Update() {
```

```
0    46            BreakoutBus.GetBus().ProcessEventsSequentially();
0    47            stateMachine.ActiveState.UpdateState();
0    48        }
     49    }
```

Breakout.Health

Summary

Class:	Breakout.Health
Assembly:	Breakout
File(s):	/home/student/SU23Guest/DIKUGames/Breakout/Health.cs
Covered lines:	31
Uncovered lines:	3
Coverable lines:	34
Total lines:	59
Line coverage:	91.1% (31 of 34)
Covered branches:	8
Total branches:	8
Branch coverage:	100% (8 of 8)
Covered methods:	4
Total methods:	5
Method coverage:	80% (4 of 5)

Metrics

Method	Branch coverage	Cyclomatic complexity	Line coverage
get_Health()	100%	1	100%
.ctor()	100%	1	100%
ProcessEvent(...)	100%	6	100%
LoseHealth()	100%	2	100%
Render()	100%	1	0%

File(s)

/home/student/SU23Guest/DIKUGames/Breakout/Health.cs

#	Line	Line coverage
	1	`using DIKUArcade.Graphics;`
	2	`using DIKUArcade.Math;`
	3	`using DIKUArcade.Events;`
	4	`namespace Breakout;`
	5	`/// <summary>`

```
        6   /// Player health.
        7   /// </summary>
        8   public class Health : IGameEventProcessor {
        9       private int health;
       10       private Text display;
       11       public int _Health {
   19  12           get => health;
       13       }
  128  14       public Health() {
   64  15           health = 3;
   64  16           display = new Text($"Lives: {health}",
   64  17               new Vec2F(0.85f, -0.275f),
   64  18               new Vec2F(0.25f, 0.35f));
   64  19           display.SetColor(new Vec3I(255, 255, 255));
   64  20           BreakoutBus.GetBus().Subscribe(GameEventType.StatusEvent, this);
   64  21       }
       22       /// <summary>
       23       /// Uses StausEvents to either lose or get health.
       24       /// </summary>
  552  25       public void ProcessEvent(GameEvent gameEvent) {
 1104  26           if (gameEvent.EventType == GameEventType.StatusEvent) {
  552  27               switch (gameEvent.Message) {
       28                   case "LOSE HEALTH":
   58  29                       LoseHealth();
   58  30                       display.SetText("Lives:" + health.ToString());
   58  31                       break;
       32                   case "GET HEALTH":
   14  33                       health += gameEvent.IntArg1;
   14  34                       display.SetText("Lives:" + health.ToString());
   14  35                       break;
       36               }
  552  37           }
  552  38       }
       39       /// <summary>
       40       /// Decrements health if health is 0 state switches to game lost.
       41       /// </summary>
   62  42       public void LoseHealth() {
   62  43           health -= 1;
   85  44           if (health <= 0) {
   23  45               health = 0;
```

```
23  46                BreakoutBus.GetBus().RegisterEvent(new GameEvent {
23  47                    EventType = GameEventType.GameStateEvent,
23  48                    Message = "CHANGE_STATE",
23  49                    StringArg1 = "GAME_LOST"
23  50                });
23  51            }
62  52        }
    53        /// <summary>
    54        /// Renders health text
    55        /// </summary>
 0  56        public void Render() {
 0  57            display.RenderText();
 0  58        }
    59    }
```

Breakout.Levels.LevelCreator

Summary

Class:	Breakout.Levels.LevelCreator
Assembly:	Breakout
File(s):	/home/student/SU23Guest/DIKUGames/Breakout/LevelLoading/LevelCreator.cs
Covered lines:	53
Uncovered lines:	0
Coverable lines:	53
Total lines:	90
Line coverage:	100% (53 of 53)
Covered branches:	11
Total branches:	12
Branch coverage:	91.6% (11 of 12)
Covered methods:	7
Total methods:	7
Method coverage:	100% (7 of 7)

Metrics

Method	Branch coverage	Cyclomatic complexity	Line coverage
get_Blocks()	100%	1	100%
get_Time()	100%	1	100%
get_HasTimer()	100%	1	100%
.ctor()	100%	1	100%
CreateLevel(...)	100%	2	100%
CreateBlocks()	100%	6	100%
InitializeTimer()	75.00%	4	100%

File(s)

/home/student/SU23Guest/DIKUGames/Breakout/LevelLoading/LevelCreator.cs

#	Line	Line coverage
	1	`using Breakout.Blocks;`
	2	`using DIKUArcade.Entities;`
	3	`using DIKUArcade.Math;`

```
      4  using System.Collections.Generic;
      5  namespace Breakout.Levels;
      6  /// <summary>
      7  /// Used to create a level from a string levelfile. Uses a level reader to read level file.
      8  /// </summary>
      9  public class LevelCreator {
     10      private string[] map;
     11      private Dictionary<string, string> meta;
     12      private Dictionary<char, string> legend;
     13      private LevelReader levelReader;
     14      private int time;
     15      private bool hasTimer;
     16      private EntityContainer<Block> blocks;
     17      public EntityContainer<Block> Blocks {
57   18          get {
57   19              return blocks;
57   20          }
     21      }
     22      public int Time {
54   23          get {
54   24              return time;
54   25          }
     26      }
     27      public bool HasTimer {
 3   28          get {
 3   29              return hasTimer;
 3   30          }
     31      }
124  32      public LevelCreator() {
62   33          this.levelReader = new LevelReader();
62   34          this.blocks = new EntityContainer<Block>(0);
62   35      }
     36      /// <summary>
     37      /// Reads a file level and creates block in Level.
     38      /// </summary>
     39      /// <param name="level">Level text file that will become the new playable level.</param>
58   40      public bool CreateLevel(string level) {
58   41          levelReader.ReadLevel(level);
114  42          if (levelReader.MapValid()) { // LevelData contains map and legend
56   43              this.map = levelReader.Map;
```

```
56    44                this.meta = levelReader.Meta;
56    45                this.legend = levelReader.Legend;
56    46                CreateBlocks();
56    47                InitializeTimer();
56    48                return true;
 2    49            } else {
 2    50                return false;
      51            }
58    52        }
      53        /// <summary>
      54        /// Uses map, legend and meta to draw blocks and apply metadata in the level.
      55        /// </summary>
56    56        private void CreateBlocks() {
      57            // map can be filled with blocks without crashing
56    58            blocks = new EntityContainer<Block>(324);
      59            // pos and extent for blocks
56    60            float x = 1f / 12f;
56    61            float y = (1f / 12f) / 2.5f;
      62            string colour;
      63            string metadata;
      64            Shape shape;
      65            Block block;
3970  66            for (int i = 0; i < map.Length - 1; i++) {
48868 67                for (int j = 0; j < map[i].Length; j++) {
15432 68                    shape = new StationaryShape(
15432 69                        new Vec2F((x * (float) j), 1.0f - (y * (float) i)),
15432 70                        new Vec2F(x, y));
19812 71                    if (legend.TryGetValue(map[i][j], out colour!)) {
4380  72                        meta.TryGetValue(map[i][j].ToString(), out metadata!);
4380  73                        block = BlockCreator.CreateBlock(shape, colour, metadata);
4380  74                        blocks.AddEntity(block);
4380  75                    }
15432 76                }
1286  77            }
56    78        }
56    79        private void InitializeTimer() {
56    80            string timeval = "";
56    81            meta.TryGetValue("Time", out timeval);
109   82            if (timeval != "" && timeval != null) {
53    83                hasTimer = true;
```

```
53    84                time = int.Parse(timeval);
56    85            } else {
 3    86                hasTimer = false;
 3    87                time = System.Int32.MaxValue;
 3    88            }
56    89        }
      90    }
```

Breakout.Levels.LevelManager

Summary

Class:	Breakout.Levels.LevelManager
Assembly:	Breakout
File(s):	/home/student/SU23Guest/DIKUGames/Breakout/LevelLoading/LevelManager.cs
Covered lines:	87
Uncovered lines:	12
Coverable lines:	99
Total lines:	159
Line coverage:	87.8% (87 of 99)
Covered branches:	26
Total branches:	30
Branch coverage:	86.6% (26 of 30)
Covered methods:	15
Total methods:	16
Method coverage:	93.7% (15 of 16)

Metrics

Method	Branch coverage	Cyclomatic complexity	Line coverage
.ctor()	100%	1	100%
get_Player()	100%	1	100%
get_Blocks()	100%	1	100%
get_Balls()	100%	1	100%
get_Powerups()	100%	1	100%
get_LevelTimer()	100%	1	100%
get_HardBalls()	100%	1	100%
NewLevel(...)	100%	1	100%
ProcessEvent(...)	100%	18	100%
EmptyLevel()	75.00%	4	85.71%
MoveBalls()	100%	2	100%
MovePowerups()	50.0%	2	60.0%
CheckCollisions()	100%	1	100%
CheckTime()	100%	2	100%
Render()	0%	2	0%
Update()	100%	1	100%

File(s)

/home/student/SU23Guest/DIKUGames/Breakout/LevelLoading/LevelManager.cs

#	Line	Line coverage
	1	`using Breakout.Blocks;`
	2	`using DIKUArcade.Events;`
	3	`using Breakout.Players;`
	4	`using DIKUArcade.Entities;`
	5	`using DIKUArcade.Graphics;`
	6	`using DIKUArcade.Math;`
	7	`using System.IO;`
	8	`using Breakout.Collisions;`
	9	`namespace Breakout.Levels;`
	10	`using Breakout.Balls;`
	11	`using Breakout.Powerups;`
	12	`/// <summary>`
	13	`/// Class that creates and manages game objects, such as blocks, balls and the player.`
	14	`/// </summary>`
	15	`public class LevelManager : IGameEventProcessor {`
	16	`    private LevelCreator levelCreator;`
	17	`    private EntityContainer<Block> blocks;`
	18	`    private EntityContainer<Ball> balls;`
	19	`    private EntityContainer<Powerup> powerups;`
	20	`    private Player player;`
58	21	`    private bool hardBalls = false;`
	22	`    private Timer levelTimer;`
	23	`    public Player Player {`
	24	`        // Used for testing`
12	25	`        get => player;`
	26	`    }`
	27	`    public EntityContainer<Block> Blocks {`
	28	`        // Used for testing`
7	29	`        get => blocks;`
	30	`    }`
	31	`    public EntityContainer<Ball> Balls {`
	32	`        // Used for testing`
10	33	`        get => balls;`
	34	`    }`
	35	`    public EntityContainer<Powerup> Powerups {`

```
            // Used for testing
13          get => powerups;
        }
        public Timer LevelTimer {
            // Used for testing
1           get => levelTimer;
        }
        public bool HardBalls {
            // Used for testing
7           get => hardBalls;
        }
116     public LevelManager() {
58          levelCreator = new LevelCreator();
58          player = new Player(
58              new DynamicShape(new Vec2F(0.425f, 0.06f), new Vec2F(0.15f, 0.04f)),
58              new Image(Path.Combine("..", "Breakout", "Assets", "Images", "player.png")));
58          balls = new EntityContainer<Ball>(18);
58          blocks = new EntityContainer<Block>(0);

58          powerups = new EntityContainer<Powerup>(10);
58          levelTimer = new Timer(new Vec2F(0.0f, -0.285f), 0);
58          BreakoutBus.GetBus().Subscribe(GameEventType.StatusEvent, this);
58      }
        /// <summary>
        /// Removes balls and creates newlevel using string levelfile
        /// </summary>
        /// <param name="level">Name of the level file that will be loaded</param>
51      public void NewLevel(string level) {
51          balls.ClearContainer(); // Reseting balls
51          levelCreator.CreateLevel(level); // Creating new level
51          blocks = levelCreator.Blocks; // Block container for new level becomes current block container
51          balls.AddEntity(BallCreator.CreateBall(new Vec2F(0.45f, 0.2f), new Vec2F(0.001f, 0.015f)));
51          levelTimer.SetTime(levelCreator.Time); // Timer for level is set
51      }
        /// <summary>
        /// Proceeses StatusEvents
        /// </summary>
767     public void ProcessEvent(GameEvent gameEvent) {
1283        if (gameEvent.EventType == GameEventType.StatusEvent) {
            switch (gameEvent.Message) {
```

```
                    case "CLEAR":
                        blocks.ClearContainer();
                        break;
                    case "NEW BALL":
                        balls.AddEntity(BallCreator.CreateBall(new Vec2F(0.45f, 0.2f), new Vec2F(0.001f, 0.015f)));
                        break;
                    case "SPAWN POWERUP":
                        Vec2F pos = (Vec2F) gameEvent.ObjectArg1;
                        powerups.AddEntity(PowerUpCreator.CreatePowerUp(pos));
                        break;
                    case "HARD BALL":
                        if (gameEvent.StringArg1 == "START") {
                            hardBalls = true;
                        } else if (gameEvent.StringArg1 == "END") {
                            hardBalls = false;
                        }
                        break;
                    case "SPLIT":
                        balls.Iterate(ball => {
                            if (balls.CountEntities() < 18) { // amount of balls that can be added is capped to 18
                                pos = ball.Shape.Position;
                                // 2 balls are added that go in different directions
                                balls.AddEntity(BallCreator.CreateBall(pos, new Vec2F(-0.0106f, 0.0106f)));
                                balls.AddEntity(BallCreator.CreateBall(pos, new Vec2F(0.0106f, 0.0106f)));
                            }
                        });
                        break;
                }
            }
        }
        /// <summary>
        /// Checks wheter if the level is containing any blocks that arent unbreakable.
        /// </summary>
        /// <returns>false if level has blocks other than unbreakable blocks, else true.</returns>
        public bool EmptyLevel() {
            foreach (Block block in blocks) {
                if (block is not Unbreakable) {
                    return false;
                }
            }
```

```
 8  116            return true;
12  117        }
14  118        private void MoveBalls() {
75  119            foreach (Ball ball in balls) {
11  120                ball.Move();
11  121            }
14  122        }
14  123        private void MovePowerups() {
42  124            foreach (Powerup powerup in powerups) {
 0  125                powerup.Move();
 0  126            }
14  127        }
14  128        private void CheckCollisions() {
14  129            PlayerCollision.Collide(balls, player);
14  130            BlockCollision.Collide(balls, blocks, hardBalls);
14  131            WallCollision.Collide(balls);
14  132            PowerUpCollision.Collide(powerups, player);
14  133        }
14  134        private void CheckTime() {
17  135            if (levelTimer.TimeLeft < 1) {
 3  136                BreakoutBus.GetBus().RegisterEvent(new GameEvent {
 3  137                    EventType = GameEventType.GameStateEvent,
 3  138                    Message = "CHANGE_STATE",
 3  139                    StringArg1 = "GAME_LOST"
 3  140                });
 3  141            }
14  142        }
 0  143        public void Render() {
 0  144            player.Render();
 0  145            blocks.RenderEntities();
 0  146            balls.RenderEntities();
 0  147            powerups.RenderEntities();
 0  148            if (levelCreator.HasTimer) {
 0  149                levelTimer.Render();
 0  150            }
 0  151        }
14  152        public void Update() {
14  153            CheckCollisions();
14  154            CheckTime();
14  155            player.Move();
```

```
14    156            MoveBalls();
14    157            MovePowerups();
14    158        }
      159    }
```

Breakout.Levels.LevelReader

Summary

Class:	Breakout.Levels.LevelReader
Assembly:	Breakout
File(s):	/home/student/SU23Guest/DIKUGames/Breakout/LevelLoading/LevelReader.cs
Covered lines:	76
Uncovered lines:	0
Coverable lines:	76
Total lines:	111
Line coverage:	100% (76 of 76)
Covered branches:	30
Total branches:	32
Branch coverage:	93.7% (30 of 32)
Covered methods:	8
Total methods:	8
Method coverage:	100% (8 of 8)

Metrics

Method	Branch coverage	Cyclomatic complexity	Line coverage
get_Legend()	100%	1	100%
get_Meta()	100%	1	100%
.ctor()	100%	1	100%
ReadLevel(...)	100%	2	100%
MapValid()	100%	6	100%
ReadMap()	83.33%	6	100%
ReadMeta()	100%	10	100%
ReadLegend()	87.50%	8	100%

File(s)

/home/student/SU23Guest/DIKUGames/Breakout/LevelLoading/LevelReader.cs

#	Line	Line coverage
	1	`using System;`
	2	`using System.Collections.Generic;`

```
      3  using System.IO;
      4  namespace Breakout.Levels;
      5  /// <summary>
      6  /// A levelReader used in Level to extract Map, Meta and Legend from a txt file.
      7  /// </summary>
      8  public class LevelReader {
      9      private string path;
     10      private string[] txtlines;
     11      private Dictionary<string, string> meta;
     12      private Dictionary<char, string> legend;
     13      public string[] Map;
     14
     15      public Dictionary<char, string> Legend {
394  16          get => legend;
     17      }
     18      public Dictionary<string, string> Meta {
379  19          get => meta;
     20      }
138  21      public LevelReader() {
 69  22          this.path = Path.Combine(Constants.MAIN_PATH, "Assets", "Levels");
 69  23      }
     24      /// <summary>
     25      /// Will try to read a level file. Reads mapdata, leveldata and metadata.
     26      /// </summary>
     27      /// <param name="level">the name of the level file that will be read</param>
     28      /// <returns>false if level file could not be read, else true.</returns>
 72  29      public bool ReadLevel(string level) {
 72  30          string txtfile = Path.Combine(path, level);
140  31          if (File.Exists(txtfile)) {
 68  32              this.txtlines = File.ReadAllLines(txtfile);
 68  33              ReadMap();
 68  34              ReadMeta();
 68  35              ReadLegend();
 68  36              return true;
  4  37          } else {
  4  38              return false;
     39          }
 72  40      }
     41      /// <summary>
     42      /// Checks if current level is valid i.e that it contains a map, legendata and metadata.
```

```
        /// </summary>
        /// <returns>true if level has map, meta and legendata, else false.</returns>
        public bool MapValid() {
            if (Map != null && Legend != null && Meta != null) {
                return true;
            } else {
                return false;
            }
        }
        private void ReadMap() {
            if (Array.IndexOf(txtlines, "Map:") == -1 ||
                Array.IndexOf(txtlines, "Map/") == -1) {
                // txt file dosent contain a start or end to Map section.
                Map = null;
            } else {
                int MapStart = Array.IndexOf(txtlines, "Map:");
                int MapEnd = Array.IndexOf(txtlines, "Map/");
                Map = new string[MapEnd - 2];
                for (int i = MapStart + 1; i < MapEnd - 1; i++) {
                    Map[i - 1] = txtlines[i];
                }
            }
        }
        private void ReadMeta() {
            if (Array.IndexOf(txtlines, "Meta:") == -1 ||
                Array.IndexOf(txtlines, "Meta/") == -1) {
                // txt file dosent contain a start or end to Meta section.
                meta = null;
            } else {
                int MetaStart = Array.IndexOf(txtlines, "Meta:");
                int MetaEnd = Array.IndexOf(txtlines, "Meta/");
                meta = new Dictionary<string, string>();
                for (int i = MetaStart + 1; i < MetaEnd; i++) {
                    string[] parts = txtlines[i].Split(": ");
                    if (parts.Length == 2) {
                        // meta section contains ": " and can be spilt in 2
                        string key = parts[0];
                        string value = parts[1];
                        if (value.Length == 1) {
                            // value is a block symbol therefore switched around
```

```
128   83                              Meta[value] = key;
257   84                          } else {
129   85                              Meta[key] = value;
129   86                          }
257   87                      }
257   88                  }
 67   89              }
 68   90          }
 68   91          private void ReadLegend() {
 68   92              if (Array.IndexOf(txtlines, "Legend:") == -1 ||
 69   93                  Array.IndexOf(txtlines, "Legend/") == -1) {
      94                  // txt file dosent contain a start or end to Legend section.
  1   95                  legend = null;
 68   96              } else {
 67   97                  int legendStart = Array.IndexOf(txtlines, "Legend:");
 67   98                  int legendEnd = Array.IndexOf(txtlines, "Legend/");
 67   99                  legend = new Dictionary<char, string>();
947  100                  for (int i = legendStart + 1; i < legendEnd; i++) {
271  101                      char symbol = txtlines[i][0];
271  102                      string imagefile = txtlines[i].Substring(3);
271  103                      string imagepath = Path.Combine(
271  104                          path.Replace(@"Levels", "Images/"), imagefile);
542  105                      if (File.Exists(imagepath)) {
271  106                          Legend[symbol] = imagefile;
271  107                      }
271  108                  }
 67  109              }
 68  110          }
     111      }
```

Breakout.Players.Player

Summary

Class:	Breakout.Players.Player
Assembly:	Breakout
File(s):	/home/student/SU23Guest/DIKUGames/Breakout/Player/Player.cs
Covered lines:	70
Uncovered lines:	6
Coverable lines:	76
Total lines:	112
Line coverage:	92.1% (70 of 76)
Covered branches:	54
Total branches:	54
Branch coverage:	100% (54 of 54)
Covered methods:	8
Total methods:	10
Method coverage:	80% (8 of 10)

Metrics

Method	Branch coverage	Cyclomatic complexity	Line coverage
.ctor(...)	100%	1	100%
get_Shape()	100%	1	0%
get_MovementSpeed()	100%	1	100%
ProcessEvent(...)	100%	46	100%
UpdateDirection()	100%	1	100%
Move()	100%	4	100%
SetMoveLeft(...)	100%	2	100%
SetMoveRight(...)	100%	2	100%
GetPosition()	100%	1	100%
Render()	100%	1	0%

File(s)

/home/student/SU23Guest/DIKUGames/Breakout/Player/Player.cs

#	Line	Line coverage

```
using DIKUArcade.Entities;
using DIKUArcade.Events;
using DIKUArcade.Graphics;
using DIKUArcade.Math;
namespace Breakout.Players;
/// <summary>
///  A player entity that can move around
/// </summary>
public class Player : Entity, IGameEventProcessor {
    private float moveLeft = 0.0f;
    private float moveRight = 0.0f;
    private float movementSpeed = 0.01f;
    private DynamicShape shape;
    public DynamicShape _Shape {
        get {
            return shape;
        }
    }
    public float MovementSpeed {
        get => movementSpeed; set => movementSpeed = value;
    }

    public Player(DynamicShape shape, IBaseImage image) : base(shape, image) {
        this.shape = base.Shape.AsDynamicShape();
        BreakoutBus.GetBus().Subscribe(GameEventType.PlayerEvent, this);
    }
    /// <summary>
    ///  processes playerEvents such as power-ups and movement events
    /// </summary>
    public void ProcessEvent(GameEvent gameEvent) {
        if (gameEvent.EventType == GameEventType.PlayerEvent) {
            switch (gameEvent.Message) {
                case "MOVE LEFT":
                    SetMoveLeft(true);
                    break;
                case "MOVE RIGHT":
                    SetMoveRight(true);
                    break;
                case "RELEASE LEFT":
                    SetMoveLeft(false);
```

84	private float moveLeft = 0.0f;
84	private float moveRight = 0.0f;
84	private float movementSpeed = 0.01f;
0	get {
0	return shape;
0	}
6	get => movementSpeed; set => movementSpeed = value;
168	public Player(DynamicShape shape, IBaseImage image) : base(shape, image) {
84	this.shape = base.Shape.AsDynamicShape();
84	BreakoutBus.GetBus().Subscribe(GameEventType.PlayerEvent, this);
84	}
331	public void ProcessEvent(GameEvent gameEvent) {
662	if (gameEvent.EventType == GameEventType.PlayerEvent) {
331	switch (gameEvent.Message) {
65	SetMoveLeft(true);
65	break;
67	SetMoveRight(true);
67	break;
62	SetMoveLeft(false);

```
62  41                      break;
    42                  case "RELEASE RIGHT":
63  43                      SetMoveRight(false);
63  44                      break;
    45                  case "SLIM JIM":
47  46                      if (gameEvent.StringArg1 == "START") {
23  47                          shape.Extent.X = 0.075f;
25  48                      } else if (gameEvent.StringArg1 == "END") {
 1  49                          shape.Extent.X = 0.15f;
 1  50                      }
24  51                      break;
    52                  case "WIDE":
53  53                      if (gameEvent.StringArg1 == "START") {
26  54                          shape.Extent.X = 0.30f;
28  55                      } else if (gameEvent.StringArg1 == "END") {
 1  56                          shape.Extent.X = 0.15f;
 1  57                      }
27  58                      break;
    59                  case "SPEED":
43  60                      if (gameEvent.StringArg1 == "START") {
21  61                          movementSpeed = 0.02f;
23  62                      } else if (gameEvent.StringArg1 == "END") {
 1  63                          movementSpeed = 0.01f;
 1  64                      }
22  65                      break;
    66                  }
331 67              }
331 68          }
257 69          private void UpdateDirection() {
257 70              shape.Direction.X = moveLeft + moveRight;
257 71          }
    72          /// <summary>
    73          ///  Moves player based on directional vector
    74          /// </summary>
39  75          public void Move() {
39  76              shape.Move();
42  77              if (shape.Position.X <= 0.0f) {
 3  78                  shape.Position.X = 0.0f;
42  79              } else if ((shape.Position.X + shape.Extent.X) >= 1.0f) {
 3  80                  shape.Position.X = 1.0f - shape.Extent.X;
```

```
3    81            }
39   82        }
127  83        private void SetMoveLeft(bool val) {
192  84            if (val) {
65   85                moveLeft = -movementSpeed;
127  86            } else {
62   87                moveLeft = 0.0f;
62   88            }
127  89            UpdateDirection();
127  90        }
130  91        private void SetMoveRight(bool val) {
197  92            if (val) {
67   93                moveRight = movementSpeed;
130  94            } else {
63   95                moveRight = 0.0f;
63   96            }
130  97            UpdateDirection();
130  98        }
     99        /// <summary>
     100       ///  Gets vector position of the player
     101       /// </summary>
16   102       public Vec2F GetPosition() {
16   103           Vec2F position = new Vec2F(shape.Position.X, shape.Position.Y);
16   104           return (position);
16   105       }
     106       /// <summary>
     107       ///  Renders the player
     108       /// </summary>
0    109       public void Render() {
0    110           RenderEntity();
0    111       }
     112   }
```

Breakout.Points

Summary

Class:	Breakout.Points
Assembly:	Breakout
File(s):	/home/student/SU23Guest/DIKUGames/Breakout/Points.cs
Covered lines:	31
Uncovered lines:	3
Coverable lines:	34
Total lines:	66
Line coverage:	91.1% (31 of 34)
Covered branches:	6
Total branches:	6
Branch coverage:	100% (6 of 6)
Covered methods:	6
Total methods:	7
Method coverage:	85.7% (6 of 7)

Metrics

Method	Branch coverage	Cyclomatic complexity	Line coverage
.ctor()	100%	1	100%
GetInstance()	100%	2	100%
ProcessEvent(...)	100%	4	100%
ResetPoints()	100%	1	100%
GetPoints()	100%	1	100%
UpdateText()	100%	1	100%
Render()	100%	1	0%

File(s)

/home/student/SU23Guest/DIKUGames/Breakout/Points.cs

#	Line	Line coverage
	1	`using DIKUArcade.Events;`
	2	`using DIKUArcade.Graphics;`
	3	`using DIKUArcade.Math;`

```
        4   namespace Breakout;
        5   /// <summary>
        6   /// In-game points to be rendered on screen.
        7   /// </summary>
        8   public class Points : IGameEventProcessor {
        9       private static Points instance = null;
 74    10       private int points = 0;
       11       private Text pointText;
       12       private Vec3I white;
148    13       public Points() {
 74    14           BreakoutBus.GetBus().Subscribe(GameEventType.StatusEvent, this);
 74    15           pointText = new Text($"Points: {points}",
 74    16               new Vec2F(0.4f, -0.285f), new Vec2F(0.25f, 0.35f));
 74    17           white = new Vec3I(255, 255, 255);
 74    18           pointText.SetColor(white);
 74    19       }
       20       /// <summary>
       21       /// Retrieves or creates and instance of points
       22       /// </summary>
360    23       public static Points GetInstance() {
434    24           if (Points.instance == null) {
 74    25               Points.instance = new Points();
 74    26           }
360    27           return Points.instance;
360    28       }
       29       /// <summary>
       30       /// Procceses statusevents to get points
       31       /// </summary>
259    32       public void ProcessEvent(GameEvent gameEvent) {
518    33           if (gameEvent.EventType == GameEventType.StatusEvent) {
259    34               switch (gameEvent.Message) {
       35                   case "GET POINTS":
 45    36                       points += gameEvent.IntArg1;
 45    37                       UpdateText();
 45    38                       break;
       39               }
259    40           }
259    41       }
       42       /// <summary>
       43       /// Resets point score
```

```
        /// </summary>
74      public void ResetPoints() {
74          Points.instance = null;
74      }
        /// <summary>
        /// Gets the point score
        /// </summary>
237     public int GetPoints() {
237         return points;
237     }
        /// <summary>
        /// Updates points text to be rendered on screen.
        /// </summary>
45      private void UpdateText() {
45          pointText.SetText($"Points: {points}");
45      }
        /// <summary>
        /// Renders Points on screen.
        /// </summary>
0       public void Render() {
0           pointText.RenderText();
0       }
    }
```

Breakout.Powerups.HardBall

Summary

Class:	Breakout.Powerups.HardBall
Assembly:	Breakout
File(s):	/home/student/SU23Guest/DIKUGames/Breakout/Powerups/HardBall.cs
Covered lines:	15
Uncovered lines:	0
Coverable lines:	15
Total lines:	27
Line coverage:	100% (15 of 15)
Covered branches:	0
Total branches:	0
Covered methods:	2
Total methods:	2
Method coverage:	100% (2 of 2)

Metrics

Method	Branch coverage	Cyclomatic complexity	Line coverage
.ctor(...)	100%	1	100%
Effect()	100%	1	100%

File(s)

/home/student/SU23Guest/DIKUGames/Breakout/Powerups/HardBall.cs

#	Line	Line coverage
	1	`using DIKUArcade.Entities;`
	2	`using DIKUArcade.Events;`
	3	`using DIKUArcade.Graphics;`
	4	`using DIKUArcade.Timers;`
	5	`using System.IO;`
	6	`namespace Breakout.Powerups;`
	7	`/// <summary>`
	8	`/// Balls destroy blocks regardless of health and don't change direction when hitting blocks.`
	9	`/// The powerup is timed and therefore temporary`

```
    /// </summary>
    public class HardBall : Powerup {
        public HardBall(DynamicShape shape) : base(shape, new Image(
            Path.Combine("..", "Breakout", "Assets", "Images", "ExtraBallPowerUp.png"))) {
        }
        public override void Effect() {
            BreakoutBus.GetBus().RegisterEvent(new GameEvent {
                EventType = GameEventType.StatusEvent,
                Message = "HARD BALL",
                StringArg1 = "START"
            });
            BreakoutBus.GetBus().RegisterTimedEvent(new GameEvent {
                EventType = GameEventType.StatusEvent,
                Message = "HARD BALL",
                StringArg1 = "END"
            }, TimePeriod.NewSeconds(10.0));
        }
    }
```

Breakout.Powerups.LifeLoss

Summary

Class:	Breakout.Powerups.LifeLoss
Assembly:	Breakout
File(s):	/home/student/SU23Guest/DIKUGames/Breakout/Powerups/Hazards/LifeLoss.cs
Covered lines:	9
Uncovered lines:	0
Coverable lines:	9
Total lines:	19
Line coverage:	100% (9 of 9)
Covered branches:	0
Total branches:	0
Covered methods:	2
Total methods:	2
Method coverage:	100% (2 of 2)

Metrics

Method	Branch coverage	Cyclomatic complexity	Line coverage
.ctor(...)	100%	1	100%
Effect()	100%	1	100%

File(s)

/home/student/SU23Guest/DIKUGames/Breakout/Powerups/Hazards/LifeLoss.cs

#	Line	Line coverage
	1	`using DIKUArcade.Entities;`
	2	`using DIKUArcade.Events;`
	3	`using DIKUArcade.Graphics;`
	4	`using System.IO;`
	5	`namespace Breakout.Powerups;`
	6	`/// <summary>`
	7	`/// When picked up, the powerup sends and event making`
	8	`/// </summary>`
	9	`public class LifeLoss : Powerup {`

```
15    10        public LifeLoss(DynamicShape shape) : base(shape, new Image(
30    11            Path.Combine("..", "Breakout", "Assets", "Images", "LoseLife.png"))) {
15    12        }
 1    13        public override void Effect() {
 1    14            BreakoutBus.GetBus().RegisterEvent(new GameEvent {
 1    15                EventType = GameEventType.StatusEvent,
 1    16                Message = "LOSE HEALTH",
 1    17            });
 1    18        }
      19    }
```

Breakout.Powerups.LifePlus

Summary

Class:	Breakout.Powerups.LifePlus
Assembly:	Breakout
File(s):	/home/student/SU23Guest/DIKUGames/Breakout/Powerups/LifePlus.cs
Covered lines:	10
Uncovered lines:	0
Coverable lines:	10
Total lines:	21
Line coverage:	100% (10 of 10)
Covered branches:	0
Total branches:	0
Covered methods:	2
Total methods:	2
Method coverage:	100% (2 of 2)

Metrics

Method	Branch coverage	Cyclomatic complexity	Line coverage
.ctor(...)	100%	1	100%
Effect()	100%	1	100%

File(s)

/home/student/SU23Guest/DIKUGames/Breakout/Powerups/LifePlus.cs

#	Line	Line coverage
	1	`using DIKUArcade.Entities;`
	2	`using DIKUArcade.Events;`
	3	`using DIKUArcade.Graphics;`
	4	`using System.IO;`
	5	`namespace Breakout.Powerups;`
	6	`/// <summary>`
	7	`/// When picked up, the powerup sends an event making health increase`
	8	`/// </summary>`
	9	`public class LifePlus : Powerup {`

```
        public LifePlus(DynamicShape shape) :
            base(shape, new Image(
                Path.Combine("..", "Breakout", "Assets", "Images", "LifePickUp.png"))) {
        }
        public override void Effect() {
            BreakoutBus.GetBus().RegisterEvent(new GameEvent {
                EventType = GameEventType.StatusEvent,
                Message = "GET HEALTH",
                IntArg1 = 1
            });
        }
    }
```

Breakout.Powerups.PlayerSpeed

Summary

Class:	Breakout.Powerups.PlayerSpeed
Assembly:	Breakout
File(s):	/home/student/SU23Guest/DIKUGames/Breakout/Powerups/PlayerSpeed.cs
Covered lines:	15
Uncovered lines:	0
Coverable lines:	15
Total lines:	27
Line coverage:	100% (15 of 15)
Covered branches:	0
Total branches:	0
Covered methods:	2
Total methods:	2
Method coverage:	100% (2 of 2)

Metrics

Method	Branch coverage	Cyclomatic complexity	Line coverage
.ctor(...)	100%	1	100%
Effect()	100%	1	100%

File(s)

/home/student/SU23Guest/DIKUGames/Breakout/Powerups/PlayerSpeed.cs

```
#  Line    Line coverage
       1    using DIKUArcade.Entities;
       2    using DIKUArcade.Events;
       3    using DIKUArcade.Graphics;
       4    using DIKUArcade.Timers;
       5    using System.IO;
       6    namespace Breakout.Powerups;
       7    /// <summary>
       8    /// When picked up, player movement speed increases by sending a PlayerEvent.
       9    /// The powerup is timed and therefore temporary
```

```
    /// </summary>
    public class PlayerSpeed : Powerup {
        public PlayerSpeed(DynamicShape shape) : base(shape, new Image(
            Path.Combine("..", "Breakout", "Assets", "Images", "DoubleSpeedPowerUp.png"))) {
        }
        public override void Effect() {
            BreakoutBus.GetBus().RegisterEvent(new GameEvent {
                EventType = GameEventType.PlayerEvent,
                Message = "SPEED",
                StringArg1 = "START"
            });
            BreakoutBus.GetBus().RegisterTimedEvent(new GameEvent {
                EventType = GameEventType.PlayerEvent,
                Message = "SPEED",
                StringArg1 = "END"
            }, TimePeriod.NewSeconds(10.0));
        }
    }
```

Breakout.Powerups.Powerup

Summary

Class:	Breakout.Powerups.Powerup
Assembly:	Breakout
File(s):	/home/student/SU23Guest/DIKUGames/Breakout/Powerups/Powerup.cs
Covered lines:	8
Uncovered lines:	0
Coverable lines:	8
Total lines:	23
Line coverage:	100% (8 of 8)
Covered branches:	2
Total branches:	2
Branch coverage:	100% (2 of 2)
Covered methods:	2
Total methods:	2
Method coverage:	100% (2 of 2)

Metrics

Method	Branch coverage	Cyclomatic complexity	Line coverage
.ctor(...)	100%	1	100%
Move()	100%	2	100%

File(s)

/home/student/SU23Guest/DIKUGames/Breakout/Powerups/Powerup.cs

#	Line	Line coverage
	1	`using DIKUArcade.Entities;`
	2	`using DIKUArcade.Graphics;`
	3	`namespace Breakout.Powerups;`
	4	`/// <summary>`
	5	`/// An abstract powerup which allows powerups to move and activate an effect.`
	6	`/// </summary>`
	7	`public abstract class Powerup : Entity {`
188	8	`    public Powerup(DynamicShape shape, IBaseImage image) : base(shape, image) {`

```
94   9        }
    10        /// <summary>
    11        /// Moves the powerup
    12        /// </summary>
 5  13        public void Move() {
 5  14            this.Shape.Move();
 6  15            if (Shape.Position.Y <= 0.0 - Shape.Extent.Y) {
 1  16                DeleteEntity();
 1  17            }
 5  18        }
    19        /// <summary>
    20        /// Activates powerup effect, by sending a GameEvent.
    21        /// </summary>
    22        public abstract void Effect();
    23    }
```

Breakout.Powerups.PowerUpCreator

Summary

Class:	Breakout.Powerups.PowerUpCreator
Assembly:	Breakout
File(s):	/home/student/SU23Guest/DIKUGames/Breakout/Powerups/PowerupCreator.cs
Covered lines:	34
Uncovered lines:	0
Coverable lines:	34
Total lines:	56
Line coverage:	100% (34 of 34)
Covered branches:	7
Total branches:	7
Branch coverage:	100% (7 of 7)
Covered methods:	2
Total methods:	2
Method coverage:	100% (2 of 2)

Metrics

Method	Branch coverage	Cyclomatic complexity	Line coverage
.cctor()	100%	1	100%
CreatePowerUp(...)	100%	7	100%

File(s)

/home/student/SU23Guest/DIKUGames/Breakout/Powerups/PowerupCreator.cs

#	Line	Line coverage
	1	`using DIKUArcade.Entities;`
	2	`using DIKUArcade.Math;`
	3	`using System;`
	4	
	5	`namespace Breakout.Powerups;`
	6	`/// <summary>`
	7	`///  Creates a Powerup`
	8	`/// </summary>`

```
        9    public static class PowerUpCreator {
 1     10        private static Vec2F extent = new Vec2F(0.03f, 0.03f);
 1     11        private static Vec2F dir = new Vec2F(0.00f, -0.01f);
       12        /// <summary>
       13        ///  Creates a random powerup
       14        /// </summary>
81     15        public static Powerup CreatePowerUp(Vec2F pos) {
81     16            Random random = new Random();
81     17            switch (random.Next(1, 8)) {
       18                case 1:
11     19                    return new LifePlus(new DynamicShape(
11     20                    pos,
11     21                    extent,
11     22                    dir));
       23                case 2:
14     24                    return new LifeLoss(new DynamicShape(
14     25                    pos,
14     26                    extent,
14     27                    dir));
       28                case 3:
12     29                    return new Wide(new DynamicShape(
12     30                    pos,
12     31                    extent,
12     32                    dir));
       33                case 4:
10     34                    return new SlimJim(new DynamicShape(
10     35                    pos,
10     36                    extent,
10     37                    dir));
       38                case 5:
 6     39                    return new PlayerSpeed(new DynamicShape(
 6     40                    pos,
 6     41                    extent,
 6     42                    dir));
       43                case 6:
15     44                    return new Split(new DynamicShape(
15     45                    pos,
15     46                    extent,
15     47                    dir));
       48                default:
```

```
13  49                  return new HardBall(new DynamicShape(
13  50                  pos,
13  51                  extent,
13  52                  dir));
    53              }
81  54          }
    55      }
    56
```

Breakout.Powerups.SlimJim

Summary

Class:	Breakout.Powerups.SlimJim
Assembly:	Breakout
File(s):	/home/student/SU23Guest/DIKUGames/Breakout/Powerups/Hazards/SlimJim.cs
Covered lines:	15
Uncovered lines:	0
Coverable lines:	15
Total lines:	27
Line coverage:	100% (15 of 15)
Covered branches:	0
Total branches:	0
Covered methods:	2
Total methods:	2
Method coverage:	100% (2 of 2)

Metrics

Method	Branch coverage	Cyclomatic complexity	Line coverage
.ctor(...)	100%	1	100%
Effect()	100%	1	100%

File(s)

/home/student/SU23Guest/DIKUGames/Breakout/Powerups/Hazards/SlimJim.cs

#	Line	Line coverage
	1	`using DIKUArcade.Entities;`
	2	`using DIKUArcade.Events;`
	3	`using DIKUArcade.Graphics;`
	4	`using DIKUArcade.Timers;`
	5	`using System.IO;`
	6	`namespace Breakout.Powerups;`
	7	`/// <summary>`
	8	`/// When picked up, player size is decreased by sending out a PlayerEvent.`
	9	`/// The powerup is timed and therefore temporary`

```
    /// </summary>
    public class SlimJim : Powerup {
        public SlimJim(DynamicShape shape) : base(shape, new Image(
            Path.Combine("..", "Breakout", "Assets", "Images", "SlimJim.png"))) {
        }
        public override void Effect() {
            BreakoutBus.GetBus().RegisterEvent(new GameEvent {
                EventType = GameEventType.PlayerEvent,
                Message = "SLIM JIM",
                StringArg1 = "START"
            });
            BreakoutBus.GetBus().RegisterTimedEvent(new GameEvent {
                EventType = GameEventType.PlayerEvent,
                Message = "SLIM JIM",
                StringArg1 = "END"
            }, TimePeriod.NewSeconds(10.0));
        }
    }
```

Breakout.Powerups.Split

Summary

Class:	Breakout.Powerups.Split
Assembly:	Breakout
File(s):	/home/student/SU23Guest/DIKUGames/Breakout/Powerups/Split.cs
Covered lines:	9
Uncovered lines:	0
Coverable lines:	9
Total lines:	19
Line coverage:	100% (9 of 9)
Covered branches:	0
Total branches:	0
Covered methods:	2
Total methods:	2
Method coverage:	100% (2 of 2)

Metrics

Method	Branch coverage	Cyclomatic complexity	Line coverage
.ctor(...)	100%	1	100%
Effect()	100%	1	100%

File(s)

/home/student/SU23Guest/DIKUGames/Breakout/Powerups/Split.cs

#	Line	Line coverage
	1	`using DIKUArcade.Entities;`
	2	`using DIKUArcade.Events;`
	3	`using DIKUArcade.Graphics;`
	4	`using System.IO;`
	5	`namespace Breakout.Powerups;`
	6	`/// <summary>`
	7	`/// Ball splits into 3 balls going different directions.`
	8	`/// </summary>`
	9	`public class Split : Powerup {`

```
16  10        public Split(DynamicShape shape) : base(shape, new Image(
32  11            Path.Combine("..", "Breakout", "Assets", "Images", "SplitPowerUp.png"))) {
16  12        }
 2  13        public override void Effect() {
 2  14            BreakoutBus.GetBus().RegisterEvent(new GameEvent {
 2  15                EventType = GameEventType.StatusEvent,
 2  16                Message = "SPLIT",
 2  17            });
 2  18        }
    19    }
```

Breakout.Powerups.Wide

Summary

Class:	Breakout.Powerups.Wide
Assembly:	Breakout
File(s):	/home/student/SU23Guest/DIKUGames/Breakout/Powerups/Wide.cs
Covered lines:	15
Uncovered lines:	0
Coverable lines:	15
Total lines:	27
Line coverage:	100% (15 of 15)
Covered branches:	0
Total branches:	0
Covered methods:	2
Total methods:	2
Method coverage:	100% (2 of 2)

Metrics

Method	Branch coverage	Cyclomatic complexity	Line coverage
.ctor(...)	100%	1	100%
Effect()	100%	1	100%

File(s)

/home/student/SU23Guest/DIKUGames/Breakout/Powerups/Wide.cs

#	Line	Line coverage
	1	`using DIKUArcade.Entities;`
	2	`using DIKUArcade.Events;`
	3	`using DIKUArcade.Graphics;`
	4	`using DIKUArcade.Timers;`
	5	`using System.IO;`
	6	`namespace Breakout.Powerups;`
	7	`/// <summary>`
	8	`/// When picked up, player size is increased by sending out a PlayerEvent.`
	9	`/// The powerup is timed and therefore temporary`

```
    /// </summary>
    public class Wide : Powerup {
        public Wide(DynamicShape shape) : base(shape, new Image(
            Path.Combine("..", "Breakout", "Assets", "Images", "WidePowerUp.png"))) {
        }
        public override void Effect() {
            BreakoutBus.GetBus().RegisterEvent(new GameEvent {
                EventType = GameEventType.PlayerEvent,
                Message = "WIDE",
                StringArg1 = "START"
            });
            BreakoutBus.GetBus().RegisterTimedEvent(new GameEvent {
                EventType = GameEventType.PlayerEvent,
                Message = "WIDE",
                StringArg1 = "END"
            }, TimePeriod.NewSeconds(10.0));
        }
    }
```

Breakout.Program

Summary

Class:	Breakout.Program
Assembly:	Breakout
File(s):	/home/student/SU23Guest/DIKUGames/Breakout/Program.cs
Covered lines:	0
Uncovered lines:	5
Coverable lines:	5
Total lines:	9
Line coverage:	0% (0 of 5)
Covered branches:	0
Total branches:	0
Covered methods:	0
Total methods:	1
Method coverage:	0% (0 of 1)

Metrics

Method	Branch coverage	Cyclomatic complexity	Line coverage
Main(...)	100%	1	0%

File(s)

/home/student/SU23Guest/DIKUGames/Breakout/Program.cs

#	Line	Line coverage
	1	`using DIKUArcade.GUI;`
	2	`namespace Breakout;`
	3	`class Program {`
0	4	`    static void Main(string[] args) {`
0	5	`            var windowArgs = new WindowArgs() {Title = "Breakout"};`
0	6	`            var game = new Game(windowArgs);`
0	7	`            game.Run();`
0	8	`        }`
	9	`}`

Breakout.States.GameLost

Summary

Class:	Breakout.States.GameLost
Assembly:	Breakout
File(s):	/home/student/SU23Guest/DIKUGames/Breakout/States/GameLost.cs
Covered lines:	74
Uncovered lines:	15
Coverable lines:	89
Total lines:	139
Line coverage:	83.1% (74 of 89)
Covered branches:	11
Total branches:	12
Branch coverage:	91.6% (11 of 12)
Covered methods:	7
Total methods:	9
Method coverage:	77.7% (7 of 9)

Metrics

Method	Branch coverage	Cyclomatic complexity	Line coverage
.ctor()	100%	1	100%
get_ActiveMenuButton	100%	1	100%
GetInstance()	100%	2	100%
InitializeGameState(	100%	1	100%
ResetState()	100%	1	100%
UpdateState()	100%	1	0%
RenderState()	100%	1	0%
HandleKeyEvent(...)	100%	2	100%
KeyPress(...)	87.50%	8	76.00%

File(s)

/home/student/SU23Guest/DIKUGames/Breakout/States/GameLost.cs

#	Line	Line coverage
	1	`using DIKUArcade.Entities;`

```
using DIKUArcade.Events;
using DIKUArcade.Graphics;
using DIKUArcade.Input;
using DIKUArcade.Math;
using DIKUArcade.State;
using System.IO;
namespace Breakout.States;
/// <summary>
///  A state for when the game is lost
/// </summary>
public class GameLost : IGameState {
    private static GameLost instance = null;
    private Points points = null!;
    private Entity background;
    private Text[] menuButtons = new Text[2];
    private Text gameOverText;
    private Text pointsText;
    private int pointsValue;
    private int activeMenuButton;
    public int ActiveMenuButton {
        get => activeMenuButton;
    }
    private const int MAIN_MENU = 0;
    private const int QUIT = 1;
    private Vec3I white = new Vec3I(255, 255, 255);
    private Vec3I red = new Vec3I(255, 0, 0);
    /// <summary>
    ///  Gets or creates an instance of the GameLost state
    /// </summary>
    public static GameLost GetInstance() {
        if (GameLost.instance == null) {
            GameLost.instance = new GameLost();
            GameLost.instance.InitializeGameState();
        }
        return GameLost.instance;
    }
    /// <summary>
    ///  Inizializes the Game state, this functions as a constructor for the state
    /// </summary>
    public void InitializeGameState() {
```

```
208  42            points = Points.GetInstance();
208  43            pointsValue = points.GetPoints();
     44
208  45            background = new Entity(
208  46                new StationaryShape(
208  47                    new Vec2F(0.0f, 0.0f),
208  48                    new Vec2F(1.0f, 1.0f)),
208  49                    new Image(Path.Combine(
208  50                    "..", "Breakout", "Assets", "Images", "SpaceBackground.png")));
208  51            gameOverText = new Text(
208  52                "Game over",
208  53                new Vec2F(0.30f, 0.17f),
208  54                new Vec2F(0.7f, 0.7f)
208  55                );
     56
208  57            pointsText = new Text(
208  58                $"Points: {pointsValue}",
208  59                new Vec2F(0.41f, 0.32f),
208  60                new Vec2F(0.4f, 0.4f)
208  61            );
     62
208  63            gameOverText.SetColor(white);
208  64            pointsText.SetColor(white);
     65
208  66            menuButtons[MAIN_MENU] = new Text(
208  67                "Main Menu",
208  68                new Vec2F(0.39f, 0.1f),
208  69                new Vec2F(0.4f, 0.4f)
208  70                );
     71
208  72            menuButtons[QUIT] = new Text(
208  73                "Quit game",
208  74                new Vec2F(0.4f, 0f),
208  75                new Vec2F(0.4f, 0.4f)
208  76                );
208  77            activeMenuButton = MAIN_MENU;
208  78            menuButtons[MAIN_MENU].SetColor(red);
208  79            menuButtons[QUIT].SetColor(white);
208  80        }
     81        /// <summary>
```

```
        /// Resets the state
        /// </summary>
205     public void ResetState() {
205         GameLost.instance = null;
205     }
        /// <summary>
        ///  Updates the state, this an empty method
        /// </summary>
  0     public void UpdateState() {
  0     }
        /// <summary>
        ///  Renders objects in the state
        /// </summary>
  0     public void RenderState() {
  0         background.RenderEntity();
  0         gameOverText.RenderText();
  0         menuButtons[QUIT].RenderText();
  0         menuButtons[MAIN_MENU].RenderText();
  0         pointsText.RenderText();
  0     }
        /// <summary>
        ///  Handles key input events such as key presses and key realising
        /// </summary>
  4     public void HandleKeyEvent(KeyboardAction action, KeyboardKey key) {
  8         if (action == KeyboardAction.KeyPress) {
  4             KeyPress(key);
  4         }
  4     }
  4     private void KeyPress(KeyboardKey key) {
  4         switch (key) {
            case KeyboardKey.Up:
  2             activeMenuButton = MAIN_MENU;
  2             menuButtons[MAIN_MENU].SetColor(red);
  2             menuButtons[QUIT].SetColor(white);
  2             break;
            case KeyboardKey.Down:
  1             activeMenuButton = QUIT;
  1             menuButtons[QUIT].SetColor(red);
  1             menuButtons[MAIN_MENU].SetColor(white);
  1             break;
```

```
                case KeyboardKey.Enter:
                    if (ActiveMenuButton == MAIN_MENU) {
                        BreakoutBus.GetBus().RegisterEvent(new GameEvent {
                            EventType = GameEventType.GameStateEvent,
                            Message = "CHANGE_STATE",
                            StringArg1 = "MAIN_MENU"
                        });
                    } else {
                        BreakoutBus.GetBus().RegisterEvent(new GameEvent {
                            EventType = GameEventType.WindowEvent,
                            Message = "CLOSE_GAME",
                            StringArg1 = "WINDOW CLOSE"
                        });
                    }
                    break;
            }
        }
    }
```

Breakout.States.GamePaused

Summary

Class:	Breakout.States.GamePaused
Assembly:	Breakout
File(s):	/home/student/SU23Guest/DIKUGames/Breakout/States/GamePaused.cs
Covered lines:	73
Uncovered lines:	8
Coverable lines:	81
Total lines:	129
Line coverage:	90.1% (73 of 81)
Covered branches:	12
Total branches:	12
Branch coverage:	100% (12 of 12)
Covered methods:	9
Total methods:	11
Method coverage:	81.8% (9 of 11)

Metrics

Method	Branch coverage	Cyclomatic complexity	Line coverage
get_Background()	100%	1	100%
.ctor()	100%	1	100%
get_PauseText()	100%	1	100%
get_ActiveMenuButton	100%	1	100%
GetInstance()	100%	2	100%
InitializeGameState(	100%	1	100%
ResetState()	100%	1	100%
UpdateState()	100%	1	0%
RenderState()	100%	1	0%
HandleKeyEvent(...)	100%	2	100%
KeyPress(...)	100%	8	100%

File(s)

/home/student/SU23Guest/DIKUGames/Breakout/States/GamePaused.cs

#	Line	Line coverage
	1	`using DIKUArcade.Entities;`
	2	`using DIKUArcade.Events;`
	3	`using DIKUArcade.Graphics;`
	4	`using DIKUArcade.Input;`
	5	`using DIKUArcade.Math;`
	6	`using DIKUArcade.State;`
	7	`using System.IO;`
	8	
	9	`namespace Breakout.States;`
	10	`/// <summary>`
	11	`///  A state for when the game is paused`
	12	`/// </summary>`
	13	`public class GamePaused : IGameState {`
	14	`    private static GamePaused instance = null;`
	15	`    private Entity background;`
	16	`    public Entity Background {`
1	17	`        get => background;`
	18	`    }`
38	19	`    private Text[] menuButtons = new Text[2];`
	20	`    private Text pauseText;`
	21	`    public Text PauseText {`
1	22	`        get => pauseText;`
	23	`    }`
	24	`    private int activeMenuButton;`
	25	`    public int ActiveMenuButton {`
5	26	`        get => activeMenuButton;`
	27	`    }`
	28	`    private const int CONTINUE = 0;`
	29	`    private const int MAIN_MENU = 1;`
38	30	`    private Vec3I white = new Vec3I(255, 255, 255);`
38	31	`    private Vec3I red = new Vec3I(255, 0, 0);`
	32	`    /// <summary>`
	33	`    ///  Gets or creates an instance of the GamePaused state`
	34	`    /// </summary>`
49	35	`    public static GamePaused GetInstance() {`
74	36	`        if (GamePaused.instance == null) {`
25	37	`            GamePaused.instance = new GamePaused();`
25	38	`            GamePaused.instance.InitializeGameState();`
25	39	`        }`

```
49  40            return GamePaused.instance;
49  41        }
    42        /// <summary>
    43        ///  Inizializes the Game state, this functions as a constructor for the state
    44        /// </summary>
30  45        public void InitializeGameState() {
30  46            background = new Entity(
30  47                new StationaryShape(
30  48                    new Vec2F(0.0f, 0.0f),
30  49                    new Vec2F(1.0f, 1.0f)),
30  50                    new Image(Path.Combine(
30  51                    "..", "Breakout", "Assets", "Images", "SpaceBackground.png")));
30  52            pauseText = new Text(
30  53                "Paused",
30  54                new Vec2F(0.375f, 0.05f),
30  55                new Vec2F(0.7f, 0.7f)
30  56                );
30  57            activeMenuButton = CONTINUE;
30  58            menuButtons[CONTINUE] = new Text(
30  59                "Continue",
30  60                new Vec2F(0.42f, 0.2f),
30  61                new Vec2F(0.4f, 0.4f)
30  62                );
30  63            menuButtons[MAIN_MENU] = new Text(
30  64                "Main Menu",
30  65                new Vec2F(0.4f, 0.1f),
30  66                new Vec2F(0.4f, 0.4f)
30  67                );
30  68            pauseText.SetColor(white);
30  69            menuButtons[CONTINUE].SetColor(red);
30  70            menuButtons[MAIN_MENU].SetColor(white);
30  71        }
    72        /// <summary>
    73        /// Resets the state
    74        /// </summary>
25  75        public void ResetState() {
25  76            GamePaused.instance = null;
25  77        }
    78        /// <summary>
    79        ///  Updates the state, this an empty method
```

```
        /// </summary>
0       public void UpdateState() {
0       }
        /// <summary>
        ///  Renders objects in the state
        /// </summary>
0       public void RenderState() {
0           background.RenderEntity();
0           pauseText.RenderText();
0           menuButtons[CONTINUE].RenderText();
0           menuButtons[MAIN_MENU].RenderText();
0       }
        /// <summary>
        ///  Handles key input events such as key presses and key realising
        /// </summary>
7       public void HandleKeyEvent(KeyboardAction action, KeyboardKey key) {
14          if (action == KeyboardAction.KeyPress) {
7               KeyPress(key);
7           }
7       }
7       private void KeyPress(KeyboardKey key) {
7           switch (key) {
                case KeyboardKey.Up:
2                   activeMenuButton = CONTINUE;
2                   menuButtons[CONTINUE].SetColor(red);
2                   menuButtons[MAIN_MENU].SetColor(white);
2                   break;
                case KeyboardKey.Down:
3                   activeMenuButton = MAIN_MENU;
3                   menuButtons[CONTINUE].SetColor(white);
3                   menuButtons[MAIN_MENU].SetColor(red);
3                   break;
                case KeyboardKey.Enter:
3                   if (activeMenuButton == CONTINUE) {
1                       BreakoutBus.GetBus().RegisterEvent(new GameEvent {
1                           EventType = GameEventType.GameStateEvent,
1                           Message = "RESUME_STATE",
1                           StringArg1 = "GAME_RUNNING"
1                       });
2                   } else {
```

```
1   120                         BreakoutBus.GetBus().RegisterEvent(new GameEvent {
1   121                             EventType = GameEventType.GameStateEvent,
1   122                             Message = "CHANGE_STATE",
1   123                             StringArg1 = "MAIN_MENU"
1   124                         });
1   125                     }
2   126                     break;
    127             }
7   128         }
    129     }
```

Breakout.States.GameRunning

Summary

Class:	Breakout.States.GameRunning
Assembly:	Breakout
File(s):	/home/student/SU23Guest/DIKUGames/Breakout/States/GameRunning.cs
Covered lines:	98
Uncovered lines:	6
Coverable lines:	104
Total lines:	167
Line coverage:	94.2% (98 of 104)
Covered branches:	36
Total branches:	36
Branch coverage:	100% (36 of 36)
Covered methods:	14
Total methods:	15
Method coverage:	93.3% (14 of 15)

Metrics

Method	Branch coverage	Cyclomatic complexity	Line coverage
.ctor()	100%	1	100%
get_Health()	100%	1	100%
get_Background()	100%	1	100%
get_Levellst()	100%	1	100%
get_Points()	100%	1	100%
get_LevelManager()	100%	1	100%
GetInstance()	100%	2	100%
InitializeGameState(	100%	1	100%
ResetState()	100%	1	100%
RenderState()	100%	1	0%
LoadLevels()	100%	6	100%
UpdateState()	100%	1	100%
HandleKeyEvent(...)	100%	4	100%
KeyPress(...)	100%	14	100%
KeyRelease(...)	100%	10	100%

File(s)

/home/student/SU23Guest/DIKUGames/Breakout/States/GameRunning.cs

#	Line	Line coverage
	1	`using Breakout.Levels;`
	2	`using DIKUArcade.Entities;`
	3	`using DIKUArcade.Events;`
	4	`using DIKUArcade.Graphics;`
	5	`using DIKUArcade.Input;`
	6	`using DIKUArcade.Math;`
	7	`using DIKUArcade.State;`
	8	`using System.IO;`
	9	`using System.Collections.Generic;`
	10	`namespace Breakout.States;`
	11	`/// <summary>`
	12	`///  A state for when the game is running.`
	13	`/// </summary>`
	14	`public class GameRunning : IGameState {`
	15	`    private static GameRunning instance = null;`
51	16	`    private LevelManager levelManager = null!;`
51	17	`    private Points points = null!;`
	18	`    private Health health;`
51	19	`    private Entity background = null!;`
	20	`    private List<string> levelslst;`
	21	`    // Public getters for testing`
	22	`    public Health Health {`
2	23	`        get => health;`
	24	`    }`
	25	`    public Entity Background {`
4	26	`        get => background;`
	27	`    }`
	28	`    public List<string> Levellst {`
279	29	`        get => levelslst;`
	30	`    }`
	31	`    public Points Points {`
4	32	`        get => points;`
	33	`    }`
	34	`    public LevelManager LevelManager {`
77	35	`        get => levelManager;`

```
        }
        /// <summary>
        ///  Gets or creates an instance of the GameRunning state
        /// </summary>
86      public static GameRunning GetInstance() {
118         if (GameRunning.instance == null) {
32              GameRunning.instance = new GameRunning();
32              GameRunning.instance.InitializeGameState();
32          }
86          return GameRunning.instance;
86      }
        /// <summary>
        ///  Inizializes the Game state, this functions as a constructor for the state
        /// </summary>
42      public void InitializeGameState() {
42          background = new Entity(
42              new StationaryShape(
42                  new Vec2F(0.0f, 0.0f),
42                  new Vec2F(1.0f, 1.0f)),
42                  new Image(Path.Combine(
42                  "..", "Breakout", "Assets", "Images", "SpaceBackground.png")));
42          levelManager = new LevelManager();
42          levelslst = new List<string>();
42          Levellst.Add("level1.txt");
42          Levellst.Add("level2.txt");
42          Levellst.Add("level3.txt");
42          Levellst.Add("level4.txt");
42          Levellst.Add("wall.txt");
42          LevelManager.NewLevel(Levellst[0]);
42          points = Points.GetInstance();
42          health = new Health();
42      }
        /// <summary>
        /// Resets the state
        /// </summary>
31      public void ResetState() {
31          GameRunning.instance = null;
31      }
        /// <summary>
        ///  Renders objects in the state
```

```
        /// </summary>
0       public void RenderState() {
0           background.RenderEntity();
0           levelManager.Render();
0           points.Render();
0           health.Render();
0       }
11      private void LoadLevels() {
15          if (levelslst.Count == 0) { // No levels left to load.
4               BreakoutBus.GetBus().RegisterEvent(new GameEvent {
4                   EventType = GameEventType.GameStateEvent,
4                   Message = "CHANGE_STATE",
4                   StringArg1 = "GAME_WON"
4               });
4               return;
12          } else if (LevelManager.EmptyLevel()) {
                // If level contains no blocks except unbreakable blocks
5               levelslst.RemoveAt(0); // Removes current level from level list
5               if (Levellst.Count > 0) // Shouldnt try to access index 0 in an empty list
4                   levelManager.NewLevel(Levellst[0]);
5           }
11      }
        /// <summary>
        ///  Updates the state and loads new levelslst when the level is empty
        /// </summary>
11      public void UpdateState() {
11          levelManager.Update();
11          LoadLevels();
11      }
        /// <summary>
        ///  Handles key input events such as key presses and key realising
        /// </summary>
22      public void HandleKeyEvent(KeyboardAction action, KeyboardKey key) {
22          switch (action) {
                case KeyboardAction.KeyPress:
16                  KeyPress(key);
16                  break;
                case KeyboardAction.KeyRelease:
6                   KeyRelease(key);
6                   break;
```

```
            }
        }
        private void KeyPress(KeyboardKey key) {
            switch (key) {
                case KeyboardKey.Left:
                case KeyboardKey.A:
                    BreakoutBus.GetBus().RegisterEvent(new GameEvent {
                        EventType = GameEventType.PlayerEvent,
                        Message = "MOVE LEFT"
                    });
                    break;
                case KeyboardKey.Right:
                case KeyboardKey.D:
                    BreakoutBus.GetBus().RegisterEvent(new GameEvent {
                        EventType = GameEventType.PlayerEvent,
                        Message = "MOVE RIGHT"
                    });
                    break;
                case KeyboardKey.Escape:
                    BreakoutBus.GetBus().RegisterEvent(new GameEvent {
                        EventType = GameEventType.GameStateEvent,
                        Message = "CHANGE_STATE",
                        StringArg1 = "GAME_PAUSED"
                    });
                    break;
                case KeyboardKey.K:
                    BreakoutBus.GetBus().RegisterEvent(new GameEvent {
                        EventType = GameEventType.StatusEvent,
                        Message = "CLEAR",
                    });
                    break;
            }
        }
        private void KeyRelease(KeyboardKey key) {
            switch (key) {
                case KeyboardKey.Left:
                case KeyboardKey.A:
                    BreakoutBus.GetBus().RegisterEvent(new GameEvent {
                        EventType = GameEventType.PlayerEvent,
                        Message = "RELEASE LEFT"
```

```
2   156                     });
2   157                     break;
    158                 case KeyboardKey.Right:
    159                 case KeyboardKey.D:
2   160                     BreakoutBus.GetBus().RegisterEvent(new GameEvent {
2   161                         EventType = GameEventType.PlayerEvent,
2   162                         Message = "RELEASE RIGHT"
2   163                     });
2   164                     break;
    165             }
6   166         }
    167     }
```

Breakout.States.GameWon

Summary

Class:	Breakout.States.GameWon
Assembly:	Breakout
File(s):	/home/student/SU23Guest/DIKUGames/Breakout/States/GameWon.cs
Covered lines:	75
Uncovered lines:	15
Coverable lines:	90
Total lines:	138
Line coverage:	83.3% (75 of 90)
Covered branches:	11
Total branches:	12
Branch coverage:	91.6% (11 of 12)
Covered methods:	7
Total methods:	9
Method coverage:	77.7% (7 of 9)

Metrics

Method	Branch coverage	Cyclomatic complexity	Line coverage
.ctor()	100%	1	100%
get_ActiveMenuButton	100%	1	100%
GetInstance()	100%	2	100%
InitializeGameState(	100%	1	100%
ResetState()	100%	1	100%
UpdateState()	100%	1	0%
RenderState()	100%	1	0%
HandleKeyEvent(...)	100%	2	100%
KeyPress(...)	87.50%	8	76.00%

File(s)

/home/student/SU23Guest/DIKUGames/Breakout/States/GameWon.cs

#	Line	Line coverage
	1	`using DIKUArcade.Entities;`

```
    2   using DIKUArcade.Events;
    3   using DIKUArcade.Graphics;
    4   using DIKUArcade.Input;
    5   using DIKUArcade.Math;
    6   using DIKUArcade.State;
    7   using System.IO;
    8   namespace Breakout.States;
    9   /// <summary>
   10   ///  A state for when the game is won
   11   /// </summary>
   12   public class GameWon : IGameState {
   13       private static GameWon instance = null;
20 14       private Points points = null!;
   15       private Entity background;
20 16       private Text[] menuButtons = new Text[2];
   17       private Text gameOverText;
   18       private Text pointsText;
   19       private int pointsValue;
   20       private int activeMenuButton;
   21       public int ActiveMenuButton {
 3 22           get => activeMenuButton;
   23       }
   24       private const int MAIN_MENU = 0;
   25       private const int QUIT = 1;
20 26       private Vec3I white = new Vec3I(255, 255, 255);
20 27       private Vec3I red = new Vec3I(255, 0, 0);
   28       /// <summary>
   29       ///  Gets or creates an instance of the GameWon state
   30       /// </summary>
31 31       public static GameWon GetInstance() {
46 32           if (GameWon.instance == null) {
15 33               GameWon.instance = new GameWon();
15 34               GameWon.instance.InitializeGameState();
15 35           }
31 36           return GameWon.instance;
31 37       }
   38       /// <summary>
   39       ///  Inizializes the Game state, this functions as a constructor for the state
   40       /// </summary>
17 41       public void InitializeGameState() {
```

```
17  42          points = Points.GetInstance();
17  43          pointsValue = points.GetPoints();
    44
17  45          background = new Entity(
17  46              new StationaryShape(
17  47                  new Vec2F(0.0f, 0.0f),
17  48                  new Vec2F(1.0f, 1.0f)),
17  49                  new Image(Path.Combine(
17  50                  "..", "Breakout", "Assets", "Images", "SpaceBackground.png")));
    51
17  52          gameOverText = new Text(
17  53              "Game Won",
17  54              new Vec2F(0.30f, 0.17f),
17  55              new Vec2F(0.7f, 0.7f)
17  56              );
17  57          pointsText = new Text(
17  58              $"Points: {pointsValue}",
17  59              new Vec2F(0.41f, 0.32f),
17  60              new Vec2F(0.4f, 0.4f)
17  61          );
17  62          menuButtons[MAIN_MENU] = new Text(
17  63              "Main Menu",
17  64              new Vec2F(0.39f, 0.1f),
17  65              new Vec2F(0.4f, 0.4f)
17  66              );
    67
17  68          menuButtons[QUIT] = new Text(
17  69              "Quit game",
17  70              new Vec2F(0.4f, 0f),
17  71              new Vec2F(0.4f, 0.4f)
17  72              );
17  73          gameOverText.SetColor(white);
17  74          pointsText.SetColor(white);
17  75          activeMenuButton = MAIN_MENU;
17  76          menuButtons[MAIN_MENU].SetColor(red);
17  77          menuButtons[QUIT].SetColor(white);
17  78          points = Points.GetInstance();
17  79      }
    80      /// <summary>
    81      /// Resets the state
```

```
        82        /// </summary>
14      83        public void ResetState() {
14      84            GameWon.instance = null;
14      85        }
        86        /// <summary>
        87        ///  Updates the state, this an empty method
        88        /// </summary>
 0      89        public void UpdateState() {
 0      90        }
        91        /// <summary>
        92        ///  Renders objects in the state
        93        /// </summary>
 0      94        public void RenderState() {
 0      95            background.RenderEntity();
 0      96            gameOverText.RenderText();
 0      97            menuButtons[QUIT].RenderText();
 0      98            menuButtons[MAIN_MENU].RenderText();
 0      99            pointsText.RenderText();
 0     100        }
       101        /// <summary>
       102        ///  Handles key input events such as key presses and key realising
       103        /// </summary>
 4     104        public void HandleKeyEvent(KeyboardAction action, KeyboardKey key) {
 8     105            if (action == KeyboardAction.KeyPress) {
 4     106                KeyPress(key);
 4     107            }
 4     108        }
 4     109        private void KeyPress(KeyboardKey key) {
 4     110            switch (key) {
       111                case KeyboardKey.Up:
 2     112                    activeMenuButton = MAIN_MENU;
 2     113                    menuButtons[MAIN_MENU].SetColor(red);
 2     114                    menuButtons[QUIT].SetColor(white);
 2     115                    break;
       116                case KeyboardKey.Down:
 1     117                    activeMenuButton = QUIT;
 1     118                    menuButtons[QUIT].SetColor(red);
 1     119                    menuButtons[MAIN_MENU].SetColor(white);
 1     120                    break;
       121                case KeyboardKey.Enter:
```

```
2   122                     if (ActiveMenuButton == MAIN_MENU) {
1   123                         BreakoutBus.GetBus().RegisterEvent(new GameEvent {
1   124                             EventType = GameEventType.GameStateEvent,
1   125                             Message = "CHANGE_STATE",
1   126                             StringArg1 = "MAIN_MENU"
1   127                         });
1   128                     } else {
0   129                         BreakoutBus.GetBus().RegisterEvent(new GameEvent {
0   130                             EventType = GameEventType.WindowEvent,
0   131                             Message = "CLOSE_GAME",
0   132                             StringArg1 = "WINDOW CLOSE"
0   133                         });
0   134                     }
1   135                     break;
    136             }
4   137         }
    138     }
```

Breakout.States.MainMenu

Summary

Class:	Breakout.States.MainMenu
Assembly:	Breakout
File(s):	/home/student/SU23Guest/DIKUGames/Breakout/States/MainMenu.cs
Covered lines:	57
Uncovered lines:	13
Coverable lines:	70
Total lines:	114
Line coverage:	81.4% (57 of 70)
Covered branches:	11
Total branches:	12
Branch coverage:	91.6% (11 of 12)
Covered methods:	8
Total methods:	10
Method coverage:	80% (8 of 10)

Metrics

Method	Branch coverage	Cyclomatic complexity	Line coverage
get_BackGround()	100%	1	100%
.ctor()	100%	1	100%
get_ActiveMenuButton	100%	1	100%
GetInstance()	100%	2	100%
InitializeGameState(	100%	1	100%
ResetState()	100%	1	100%
UpdateState()	100%	1	0%
RenderState()	100%	1	0%
HandleKeyEvent(...)	100%	2	100%
KeyPress(...)	87.50%	8	76.00%

File(s)

/home/student/SU23Guest/DIKUGames/Breakout/States/MainMenu.cs

#	Line	Line coverage

```
using DIKUArcade.Entities;
using DIKUArcade.Events;
using DIKUArcade.Graphics;
using DIKUArcade.Input;
using DIKUArcade.Math;
using DIKUArcade.State;
using System.IO;
namespace Breakout.States;
/// <summary>
///  A main menu state
/// </summary>
public class MainMenu : IGameState {
    private static MainMenu instance = null;
    private Entity backGround;
    public Entity BackGround {
        get => backGround;
    }
    private Text[] menuButtons = new Text[2];
    private int activeMenuButton;
    public int ActiveMenuButton {
        get => activeMenuButton;
    }
    private const int NEW_GAME = 0;
    private const int QUIT = 1;
    private Vec3I white = new Vec3I(255, 255, 255);
    private Vec3I red = new Vec3I(255, 0, 0);
    /// <summary>
    ///  Gets or creates an instance of the MainMenu state
    /// </summary>
    public static MainMenu GetInstance() {
        if (MainMenu.instance == null) {
            MainMenu.instance = new MainMenu();
            MainMenu.instance.InitializeGameState();
        }
        return MainMenu.instance;
    }
    /// <summary>
    ///  Inizializes the Game state, this functions as a constructor for the state
    /// </summary>
    public void InitializeGameState() {
```

1	16
76	18
5	21
76	25
76	26
162	30
226	31
64	32
64	33
64	34
162	35
162	36
68	40

```
68  41            backGround = new Entity(
68  42                new StationaryShape(
68  43                    new Vec2F(0.0f, 0.0f),
68  44                    new Vec2F(1.0f, 1.0f)),
68  45                    new Image(Path.Combine(
68  46                    "..", "Breakout", "Assets", "Images", "BreakoutTitleScreen.png")));
68  47            menuButtons[NEW_GAME] = new Text("New Game",
68  48                new Vec2F(0.375f, 0.2f),
68  49                new Vec2F(0.4f, 0.4f));
68  50            menuButtons[QUIT] = new Text("Quit",
68  51                new Vec2F(0.46f, 0.1f),
68  52                new Vec2F(0.4f, 0.4f));
68  53            activeMenuButton = NEW_GAME;
68  54            menuButtons[NEW_GAME].SetColor(red);
68  55            menuButtons[QUIT].SetColor(white);
68  56            Points.GetInstance().ResetPoints();
68  57        }
    58        /// <summary>
    59        /// Resets the state
    60        /// </summary>
63  61        public void ResetState() {
63  62            MainMenu.instance = null;
63  63        }
    64        /// <summary>
    65        ///  Updates the state, this an empty method
    66        /// </summary>
 0  67        public void UpdateState() {
 0  68        }
    69        /// <summary>
    70        ///  Renders objects in the state
    71        /// </summary>
 0  72        public void RenderState() {
 0  73            backGround.RenderEntity();
 0  74            menuButtons[NEW_GAME].RenderText();
 0  75            menuButtons[QUIT].RenderText();
 0  76        }
    77        /// <summary>
    78        ///  Handles key input events such as key presses and key realising
    79        /// </summary>
 5  80        public void HandleKeyEvent(KeyboardAction action, KeyboardKey key) {
```

```
10  81            if (action == KeyboardAction.KeyPress) {
 5  82                KeyPress(key);
 5  83            }
 5  84        }
 5  85        private void KeyPress(KeyboardKey key) {
 5  86            switch (key) {
    87                case KeyboardKey.Up:
 2  88                    activeMenuButton = NEW_GAME;
 2  89                    menuButtons[NEW_GAME].SetColor(red);
 2  90                    menuButtons[QUIT].SetColor(white);
 2  91                    break;
    92                case KeyboardKey.Down:
 2  93                    activeMenuButton = QUIT;
 2  94                    menuButtons[NEW_GAME].SetColor(white);
 2  95                    menuButtons[QUIT].SetColor(red);
 2  96                    break;
    97                case KeyboardKey.Enter:
 2  98                    if (activeMenuButton == NEW_GAME) {
 1  99                        BreakoutBus.GetBus().RegisterEvent(new GameEvent {
 1 100                            EventType = GameEventType.GameStateEvent,
 1 101                            Message = "CHANGE_STATE",
 1 102                            StringArg1 = "GAME_RUNNING"
 1 103                        });
 1 104                    } else {
 0 105                        BreakoutBus.GetBus().RegisterEvent(new GameEvent {
 0 106                            EventType = GameEventType.WindowEvent,
 0 107                            Message = "CLOSE_GAME",
 0 108                            StringArg1 = "WINDOW CLOSE"
 0 109                        });
 0 110                    }
 1 111                    break;
   112            }
 5 113        }
   114    }
```

Breakout.States.StateMachine

Summary

Class:	Breakout.States.StateMachine
Assembly:	Breakout
File(s):	/home/student/SU23Guest/DIKUGames/Breakout/States/StateMachine.cs
Covered lines:	29
Uncovered lines:	0
Coverable lines:	29
Total lines:	54
Line coverage:	100% (29 of 29)
Covered branches:	12
Total branches:	12
Branch coverage:	100% (12 of 12)
Covered methods:	4
Total methods:	4
Method coverage:	100% (4 of 4)

Metrics

Method	Branch coverage	Cyclomatic complexity	Line coverage
get_ActiveState()	100%	1	100%
.ctor()	100%	1	100%
ProcessEvent(...)	100%	6	100%
SwitchState(...)	100%	6	100%

File(s)

/home/student/SU23Guest/DIKUGames/Breakout/States/StateMachine.cs

#	Line	Line coverage
	1	`using DIKUArcade.Events;`
	2	`using DIKUArcade.State;`
	3	`namespace Breakout.States;`
	4	`/// <summary>`
	5	`///  A StateMachine that changes between IGameStates`
	6	`/// </summary>`

```
      7  public class StateMachine : IGameEventProcessor {
      8      public IGameState ActiveState {
   1083  9          get; private set;
     10      }
     62 11      public StateMachine() {
     31 12          BreakoutBus.GetBus().Subscribe(GameEventType.GameStateEvent, this);
     31 13          ActiveState = MainMenu.GetInstance();
     31 14      }
     15      /// <summary>
     16      ///  Recieves GameStateEvents and can either resume a state or create a new state.
     17      /// </summary>
    357 18      public void ProcessEvent(GameEvent gameEvent) {
    714 19          if (gameEvent.EventType == GameEventType.GameStateEvent) {
    357 20              switch (gameEvent.Message) {
     21                  case ("RESUME_STATE"):
     20 22                      SwitchState(StateTransformer.TransformStringToState(gameEvent.StringArg1));
     20 23                      break;
     24                  case ("CHANGE_STATE"): // creates a new state
     25                      // Resumes a state and makes it the ActiveState
    337 26                      SwitchState(StateTransformer.TransformStringToState(gameEvent.StringArg1));
     27                      // Resets the state
    337 28                      ActiveState.ResetState();
     29                      // GetInstance() of ActiveState which is null, therefore it's initialized
    337 30                      SwitchState(StateTransformer.TransformStringToState(gameEvent.StringArg1));
    337 31                      break;
     32              }
    357 33          }
    357 34      }
    694 35      private void SwitchState(GameStateType stateType) {
    694 36          switch (stateType) {
     37              case GameStateType.GameRunning:
     82 38                  ActiveState = GameRunning.GetInstance();
     82 39                  break;
     40              case GameStateType.GamePaused:
     48 41                  ActiveState = GamePaused.GetInstance();
     48 42                  break;
     43              case GameStateType.MainMenu:
    126 44                  ActiveState = MainMenu.GetInstance();
    126 45                  break;
     46              case GameStateType.GameLost:
```

```
410    47                    ActiveState = GameLost.GetInstance();
410    48                    break;
       49                case GameStateType.GameWon:
 28    50                    ActiveState = GameWon.GetInstance();
 28    51                    break;
       52            }
694    53        }
       54    }
```

Breakout.States.StateTransformer

Summary

Class:	Breakout.States.StateTransformer
Assembly:	Breakout
File(s):	/home/student/SU23Guest/DIKUGames/Breakout/States/StateTransformer.cs
Covered lines:	9
Uncovered lines:	0
Coverable lines:	9
Total lines:	27
Line coverage:	100% (9 of 9)
Covered branches:	10
Total branches:	10
Branch coverage:	100% (10 of 10)
Covered methods:	1
Total methods:	1
Method coverage:	100% (1 of 1)

Metrics

Method	Branch coverage	Cyclomatic complexity	Line coverage
TransformStringToSta	100%	10	100%

File(s)

/home/student/SU23Guest/DIKUGames/Breakout/States/StateTransformer.cs

```
#  Line    Line coverage
       1    using System;
       2    namespace Breakout.States;
       3    /// <summary>
       4    ///  A class used to transform strings into state tyopes.
       5    /// </summary>
       6    public class StateTransformer {
       7        /// <summary>
       8        ///  Transforms strings into state types.
       9        /// </summary>
```

```
699    10        public static GameStateType TransformStringToState(string state) {
699    11            switch (state) {
       12                case "GAME_RUNNING":
 83    13                    return GameStateType.GameRunning;
       14                case "GAME_PAUSED":
 49    15                    return GameStateType.GamePaused;
       16                case "MAIN_MENU":
127    17                    return GameStateType.MainMenu;
       18                case "GAME_LOST":
411    19                    return GameStateType.GameLost;
       20                case "GAME_WON":
 28    21                    return GameStateType.GameWon;
       22                default:
  1    23                    throw new ArgumentException("Invalid GameStateType string");
       24            }
698    25        }
       26    }
       27
```

Breakout.Timer

Summary

Class:	Breakout.Timer
Assembly:	Breakout
File(s):	/home/student/SU23Guest/DIKUGames/Breakout/Timer.cs
Covered lines:	36
Uncovered lines:	0
Coverable lines:	36
Total lines:	63
Line coverage:	100% (36 of 36)
Covered branches:	6
Total branches:	6
Branch coverage:	100% (6 of 6)
Covered methods:	6
Total methods:	6
Method coverage:	100% (6 of 6)

Metrics

Method	Branch coverage	Cyclomatic complexity	Line coverage
get_TimeLeft()	100%	1	100%
.ctor(...)	100%	1	100%
SetTime(...)	100%	1	100%
UpdateTime()	100%	2	100%
UpdateText()	100%	4	100%
Render()	100%	1	100%

File(s)

/home/student/SU23Guest/DIKUGames/Breakout/Timer.cs

#	Line	Line coverage
	1	`using DIKUArcade.Timers;`
	2	`using DIKUArcade.Graphics;`
	3	`using DIKUArcade.Math;`
	4	`namespace Breakout;`

```
     5  public class Timer {
     6      private int timeLeft;
     7      private Text timerText;
     8      private int timeElapsed;
     9      private int previousTime;
    10      private int n;
    11      private Vec2F position;
    12      private Vec3I white;
    13      public int TimeLeft {
24  14          get => timeLeft;
    15      }
122 16      public Timer(Vec2F pos, int init) {
61  17          timeLeft = init;
61  18          position = pos;
61  19          white = new Vec3I(255, 255, 255);
61  20          timerText = new Text($"Time: {timeLeft}s",
61  21          position, new Vec2F(0.25f, 0.35f));
61  22          timerText.SetColor(white);
61  23          timeElapsed = 0;
61  24          n = 0;
61  25      }
    26      /// <summary>
    27      /// Sets the time left to an input value
    28      /// </summary>
54  29      public void SetTime(int s) {
54  30          timeLeft = s;
54  31      }
    32      /// <summary>
    33      /// Updates the time and decrements the amount of seconds if a second has passed
    34      /// </summary>
2   35      private void UpdateTime() {
2   36          previousTime = timeLeft;
2   37          timeElapsed = (int) StaticTimer.GetElapsedMilliseconds();
4   38          if (n + 1000 < timeElapsed) {
2   39              timeLeft--;
2   40              n = (int) StaticTimer.GetElapsedMilliseconds();
2   41          }
2   42      }
    43      /// <summary>
    44      /// Updates the text for the timer.
```

```
        /// </summary>
3       private void UpdateText() {
5           if (timeLeft > 0) {
2               UpdateTime();
4               if (previousTime != timeLeft) {
2                   timerText.SetText($"Time: {timeLeft}s");
2               }
3           } else {
1               timerText.SetText("");
1           }
3       }
        /// <summary>
        /// Renders the timer on the screen.
        /// </summary>
3       public void Render() {
3           UpdateText();
3           timerText.RenderText();
3       }
    }
```

DIKUArcade.DIKUGame

Summary

Class:	DIKUArcade.DIKUGame
Assembly:	DIKUArcade
File(s):	/home/student/SU23Guest/DIKUGames/DIKUArcade/DIKUArcade/DIKUGame.cs
Covered lines:	0
Uncovered lines:	32
Coverable lines:	32
Total lines:	73
Line coverage:	0% (0 of 32)
Covered branches:	0
Total branches:	8
Branch coverage:	0% (0 of 8)
Covered methods:	0
Total methods:	3
Method coverage:	0% (0 of 3)

Metrics

Method	Branch coverage	Cyclomatic complexity	Line coverage
get_Timestep()	100%	1	0%
.ctor(...)	100%	1	0%
Run()	0%	8	0%

File(s)

/home/student/SU23Guest/DIKUGames/DIKUArcade/DIKUArcade/DIKUGame.cs

#	Line	Line coverage
	1	`using System;`
	2	`using DIKUArcade.GUI;`
	3	`using DIKUArcade.Timers;`
	4	
	5	`namespace DIKUArcade {`
	6	`    /// <summary>`
	7	`    /// Abstract base class for any DIKUArcade game.`

```
        8         /// </summary>
        9         public abstract class DIKUGame {
       10             protected Window window;
       11             private GameTimer gameTimer;
       12
       13             /// <summary>
       14             /// The exact amount of captured updates in the last second.
       15             /// Can be used for framerate independent calculations.
       16             /// </summary>
  0    17             public static int Timestep { get; private set; }
       18
  0    19             public DIKUGame(WindowArgs windowArgs) {
  0    20                 window = new Window(windowArgs);
  0    21             }
       22
       23             /// <summary>
       24             /// Override this method to update game logic.
       25             /// </summary>
       26             public abstract void Update();
       27
       28             /// <summary>
       29             /// Override this method to render game entities.
       30             /// </summary>
       31             public abstract void Render();
       32
       33             /// <summary>
       34             /// Enter the game loop and run the game.
       35             /// This method will never return.
       36             /// </summary>
  0    37             public void Run() {
  0    38                 System.Console.WriteLine("Game.Run()");
  0    39                 gameTimer = new GameTimer(30, 30);
       40
       41                 try
  0    42                 {
  0    43                     while (window.IsRunning()) {
  0    44                         gameTimer.MeasureTime();
  0    45                         window.PollEvents();
       46
  0    47                         while (gameTimer.ShouldUpdate()) {
```

```
0    48                            Update();
0    49                        }
     50
0    51                        if (gameTimer.ShouldRender()) {
0    52                            window.Clear();
0    53                            Render();
0    54                            window.SwapBuffers();
0    55                        }
     56
0    57                        if (gameTimer.ShouldReset()) {
0    58                            Timestep = gameTimer.CapturedUpdates;
0    59                        }
0    60                    }
     61
0    62                    window.DestroyWindow();
0    63                }
0    64                catch(Exception ex) {
0    65                    Console.WriteLine("DIKUArcade.DIKUGame caught an exception. See message below:" + Environment.NewLine);
0    66                    Console.WriteLine(ex);
     67
0    68                    Console.WriteLine(Environment.NewLine + "Terminating program...");
0    69                    Environment.Exit(1);
0    70                }
0    71            }
     72        }
     73    }
```

DIKUArcade.Entities.DynamicShape

Summary

Class:	DIKUArcade.Entities.DynamicShape
Assembly:	DIKUArcade
File(s):	ome/student/SU23Guest/DIKUGames/DIKUArcade/DIKUArcade/Entities/DynamicShape.cs
Covered lines:	21
Uncovered lines:	7
Coverable lines:	28
Total lines:	51
Line coverage:	75% (21 of 28)
Covered branches:	0
Total branches:	0
Covered methods:	5
Total methods:	7
Method coverage:	71.4% (5 of 7)

Metrics

Method	Branch coverage	Cyclomatic complexity	Line coverage
.ctor(...)	100%	1	100%
.ctor(...)	100%	1	0%
.ctor(...)	100%	1	100%
.ctor(...)	100%	1	100%
ChangeDirection(...)	100%	1	100%
Move()	100%	1	100%
op_Explicit(...)	100%	1	0%

File(s)

ome/student/SU23Guest/DIKUGames/DIKUArcade/DIKUArcade/Entities/DynamicShape.cs

#	Line	Line coverage

```
1    using DIKUArcade.Math;
2
3    namespace DIKUArcade.Entities {
4        public class DynamicShape : Shape {
```

```
        5          /// <summary>
        6          /// Only dynamic entities carry a direction vector.
        7          /// </summary>
        8          public Vec2F Direction;
        9
   6   10          public DynamicShape(float posX, float posY, float width, float height) {
   3   11              Position = new Vec2F(posX, posY);
   3   12              Direction = new Vec2F();
   3   13              Extent = new Vec2F(width, height);
   3   14          }
       15
       16          public DynamicShape(float posX, float posY, float width, float height,
   0   17              float dirX, float dirY) : this(posX, posY, width, height) {
   0   18              Direction.X = dirX;
   0   19              Direction.Y = dirY;
   0   20          }
       21
 168   22          public DynamicShape(Vec2F pos, Vec2F extent) {
  84   23              Position = pos;
  84   24              Extent = extent;
  84   25              Direction = new Vec2F(0f, 0f); // init 0 to avoid problems
  84   26          }
       27
 780   28          public DynamicShape(Vec2F pos, Vec2F extent, Vec2F dir) {
 390   29              Position = pos;
 390   30              Extent = extent;
 390   31              Direction = dir;
 390   32          }
       33
   7   34          public void ChangeDirection(Vec2F dir) {
   7   35              Direction = dir;
   7   36          }
       37
       38
       39          /// <summary>
       40          /// Overrides the default Shape.Move() method to add
       41          /// this object's direction to its position.
       42          /// </summary>
 122   43          public override void Move() {
 122   44              Position += Direction;
```

```
122    45            }
       46
  0    47            public static explicit operator StationaryShape(DynamicShape obj) {
  0    48                return new StationaryShape(obj.Position, obj.Extent);
  0    49            }
       50        }
       51    }
```

DIKUArcade.Entities.Entity

Summary

Class:	DIKUArcade.Entities.Entity
Assembly:	DIKUArcade
File(s):	/home/student/SU23Guest/DIKUGames/DIKUArcade/DIKUArcade/Entities/Entity.cs
Covered lines:	13
Uncovered lines:	6
Coverable lines:	19
Total lines:	40
Line coverage:	68.4% (13 of 19)
Covered branches:	0
Total branches:	0
Covered methods:	5
Total methods:	7
Method coverage:	71.4% (5 of 7)

Metrics

Method	Branch coverage	Cyclomatic complexity	Line coverage
get_Shape()	100%	1	100%
get_Image()	100%	1	100%
.ctor(...)	100%	1	100%
DeleteEntity()	100%	1	100%
IsDeleted()	100%	1	100%
RenderEntity()	100%	1	0%
RenderEntity(...)	100%	1	0%

File(s)

/home/student/SU23Guest/DIKUGames/DIKUArcade/DIKUArcade/Entities/Entity.cs

```
#  Line   Line coverage
     1    using DIKUArcade.Graphics;
     2
     3    namespace DIKUArcade.Entities {
     4        public class Entity {
```

```
6460    5            public Shape Shape { get; set; }
5251    6            public IBaseImage Image { get; set; }
        7
        8            private bool isDeleted;
        9
10502   10           public Entity(Shape shape, IBaseImage image) {
5251    11               isDeleted = false;
5251    12               Shape = shape;
5251    13               Image = image;
5251    14           }
        15
        16           /// <summary>
        17           /// Make an Entity as ready for being deleted.
        18           /// This functionality is needed for the EntityContainer class.
        19           /// </summary>
31      20           public void DeleteEntity() {
31      21               isDeleted = true;
31      22           }
        23
        24           /// <summary>
        25           /// Check if this Entity has been marked as ready for being deleted.
        26           /// This functionality is needed for the EntityContainer class.
        27           /// </summary>
1375    28           public bool IsDeleted() {
1375    29               return isDeleted;
1375    30           }
        31
0       32           public void RenderEntity() {
0       33               Image.Render(Shape);
0       34           }
        35
0       36           public void RenderEntity(Camera camera) {
0       37               Image.Render(Shape, camera);
0       38           }
        39       }
        40   }
```

DIKUArcade.Entities.EntityContainer

Summary

Class:	DIKUArcade.Entities.EntityContainer
Assembly:	DIKUArcade
File(s):	/student/SU23Guest/DIKUGames/DIKUArcade/DIKUArcade/Entities/EntityContainer.cs
Covered lines:	0
Uncovered lines:	63
Coverable lines:	63
Total lines:	132
Line coverage:	0% (0 of 63)
Covered branches:	0
Total branches:	10
Branch coverage:	0% (0 of 10)
Covered methods:	0
Total methods:	16
Method coverage:	0% (0 of 16)

Metrics

Method	Branch coverage	Cyclomatic complexity	Line coverage
.ctor(...)	100%	1	0%
.ctor()	100%	1	0%
AddStationaryEntity(	100%	1	0%
AddDynamicEntity(...	100%	1	0%
Iterate(...)	0%	6	0%
RenderEntities()	0%	2	0%
RenderEntities(...)	0%	2	0%
ClearContainer()	100%	1	0%
CountEntities()	100%	1	0%
System.Collections.I	100%	1	0%
GetEnumerator()	100%	1	0%
.ctor(...)	100%	1	0%
MoveNext()	100%	1	0%
Reset()	100%	1	0%
System.Collections.I	100%	1	0%
get_Current()	100%	1	0%

File(s)

/student/SU23Guest/DIKUGames/DIKUArcade/DIKUArcade/Entities/EntityContainer.cs

#	Line	Line coverage
	1	`using System;`
	2	`using System.Collections;`
	3	`using System.Collections.Generic;`
	4	`using System.Collections.ObjectModel;`
	5	`using DIKUArcade.Graphics;`
	6	
	7	`namespace DIKUArcade.Entities {`
	8	`    public class EntityContainer : IEnumerable {`
	9	`        private List<Entity> entities;`
	10	
0	11	`        public EntityContainer(int size) {`
0	12	`            entities = new List<Entity>(size);`
0	13	`        }`
	14	
0	15	`        public EntityContainer() : this(50) { }`
	16	
0	17	`        public void AddStationaryEntity(StationaryShape ent, IBaseImage img) {`
0	18	`            entities.Add(new Entity(ent, img));`
0	19	`        }`
	20	
0	21	`        public void AddDynamicEntity(DynamicShape ent, IBaseImage img) {`
0	22	`            entities.Add(new Entity(ent, img));`
0	23	`        }`
	24	
	25	`        /// <summary>`
	26	`        /// Delegate method for iterating through an EntityContainer.`
	27	`        /// This function should return true if the Entity should be`
	28	`        /// removed from the EntityContainer.`
	29	`        /// </summary>`
	30	`        /// <param name="entity"></param>`
	31	`        public delegate void IteratorMethod(Entity entity);`
	32	
	33	`        /// <summary>Iterate through all Entities in this EntityContainer.</summary>`
	34	`        /// <remarks>This method can modify objects during iteration!`
	35	`        /// If this functionality is undesired, iterate then through this`

```
        /// EntityContainer using a 'foreach'-loop (from IEnumerable).</remarks>
0       public void Iterate(IteratorMethod iterator) {
0           var count = entities.Count;
0           var newList = new List<Entity>(count);

            // iterate through entities
0           for (int i = 0; i < count; i++) {
0               iterator(entities[i]);
0           }

            // keep Entities that have not been marked for deletion during iteration
0           foreach (var entity in entities) {
0               if (!entity.IsDeleted()) {
0                   newList.Add(entity);
0               }
0           }
0           entities = newList;
0       }

        /// <summary>
        /// Render all entities in this EntityContainer
        /// </summary>
0       public void RenderEntities() {
0           foreach (Entity entity in entities) {
0               entity.Image.Render(entity.Shape);
0           }
0       }

        /// <summary>
        /// Render all entities in this EntityContainer
        /// </summary>
0       public void RenderEntities(Camera camera) {
0           foreach (Entity entity in entities) {
0               entity.Image.Render(entity.Shape, camera);
0           }
0       }

        /// <summary>
        /// Remove all entities from this container
        /// </summary>
```

```
0    76            public void ClearContainer() {
0    77                entities.Clear();
0    78            }
     79
     80            /// <summary>
     81            /// Count the number of entities in the EntityContainer
     82            /// </summary>
0    83            public int CountEntities() {
0    84                return entities.Count;
0    85            }
     86
     87            // IEnumerable interface:
     88            #region IEnumerable
     89
     90            IEnumerator IEnumerable.GetEnumerator()
0    91            {
0    92                return GetEnumerator();
0    93            }
     94
0    95            public IEnumerator GetEnumerator() {
0    96                return new EntityContainerEnum(entities);
0    97            }
     98
     99            private class EntityContainerEnum : IEnumerator {
    100                private ReadOnlyCollection<Entity> entities;
0   101                private int position = -1;
    102
0   103                public EntityContainerEnum(List<Entity> entities) {
0   104                    this.entities = entities.AsReadOnly();
0   105                }
    106
0   107                public bool MoveNext() {
0   108                    position++;
0   109                    return position < entities.Count;
0   110                }
    111
0   112                public void Reset() {
0   113                    position = -1;
0   114                }
    115
```

```
0   116                 object IEnumerator.Current => Current;
    117
    118                 public Entity Current {
0   119                     get {
0   120                         try {
0   121                             return entities[position];
0   122                         } catch (IndexOutOfRangeException) {
0   123                             throw new InvalidOperationException();
    124                         }
0   125                     }
    126                 }
    127             }
    128
    129             #endregion
    130
    131         }
    132     }
```

DIKUArcade.Entities.EntityContainer<T>

Summary

Class:	DIKUArcade.Entities.EntityContainer<T>
Assembly:	DIKUArcade
File(s):	student/SU23Guest/DIKUGames/DIKUArcade/DIKUArcade/Entities/EntityContainerT.cs
Covered lines:	39
Uncovered lines:	12
Coverable lines:	51
Total lines:	113
Line coverage:	76.4% (39 of 51)
Covered branches:	6
Total branches:	8
Branch coverage:	75% (6 of 8)
Covered methods:	11
Total methods:	15
Method coverage:	73.3% (11 of 15)

Metrics

Method	Branch coverage	Cyclomatic complexity	Line coverage
.ctor(...)	100%	1	100%
.ctor()	100%	1	0%
AddEntity(...)	100%	1	100%
Iterate(...)	100%	6	100%
RenderEntities()	0%	2	0%
ClearContainer()	100%	1	100%
CountEntities()	100%	1	100%
System.Collections.I	100%	1	0%
GetEnumerator()	100%	1	100%
.ctor(...)	100%	1	100%
MoveNext()	100%	1	100%
Reset()	100%	1	0%
System.IDisposable.D	100%	1	100%
System.Collections.I	100%	1	100%
get_Current()	100%	1	100%

File(s)

student/SU23Guest/DIKUGames/DIKUArcade/DIKUArcade/Entities/EntityContainerT.cs

#	Line	Line coverage
	1	`using System;`
	2	`using System.Collections;`
	3	`using System.Collections.Generic;`
	4	`using System.Collections.ObjectModel;`
	5	`using DIKUArcade.Graphics;`
	6	
	7	`namespace DIKUArcade.Entities {`
	8	`    public sealed class EntityContainer<T> : IEnumerable where T: Entity {`
	9	`        private List<T> entities;`
	10	
644	11	`        public EntityContainer(int size) {`
322	12	`            entities = new List<T>(size);`
322	13	`        }`
	14	
0	15	`        public EntityContainer() : this(50) { }`
	16	
4739	17	`        public void AddEntity(T obj) {`
4739	18	`            entities.Add(obj);`
4739	19	`        }`
	20	
	21	`        /// <summary>`
	22	`        /// Delegate method for iterating through an EntityContainer.`
	23	`        /// This function should return true if the object should be`
	24	`        /// removed from the EntityContainer.`
	25	`        /// </summary>`
	26	`        /// <param name="obj">Generic object of type T</param>`
	27	`        public delegate void IteratorMethod(T obj);`
	28	
	29	`        /// <summary>Iterate through all objects in this EntityContainer.</summary>`
	30	`        /// <remarks>This method can modify objects during iteration!`
	31	`        /// If this functionality is undesired, iterate then through this`
	32	`        /// EntityContainer using a 'foreach'-loop (from IEnumerable).</remarks>`
193	33	`        public void Iterate(IteratorMethod iterator) {`
193	34	`            var count = entities.Count;`
193	35	`            var newList = new List<T>(count);`

```
                
                // iterate through entities
3929            for (int i = 0; i < count; i++) {
1181                iterator(entities[i]);
1181            }
                
                // keep Entities that have not been marked for deletion during iteration
4644            foreach (var obj in entities) {
2690                if (!obj.IsDeleted()) {
1335                    newList.Add(obj);
1335                }
1355            }
 193            entities = newList;
 193        }
                
            /// <summary>
            /// Render all entities in this EntityContainer
            /// </summary>
   0        public void RenderEntities() {
   0            foreach (var obj in entities) {
   0                obj.Image.Render(obj.Shape);
   0            }
   0        }
                
            /// <summary>
            /// Remove all entities from this container
            /// </summary>
 207        public void ClearContainer() {
 207            entities.Clear();
 207        }
                
            /// <summary>
            /// Count the number of entities in the EntityContainer
            /// </summary>
 201        public int CountEntities() {
 201            return entities.Count;
 201        }
                
            // IEnumerable interface:
            #region IEnumerable
```

```
        76
        77            IEnumerator IEnumerable.GetEnumerator()
 0      78            {
 0      79                return GetEnumerator();
 0      80            }
        81
42      82            public IEnumerator GetEnumerator() {
42      83                return new EntityContainerEnum(entities);
42      84            }
        85
        86            private class EntityContainerEnum : IEnumerator<T> {
        87                private ReadOnlyCollection<T> entities;
42      88                private int position = -1;
        89
84      90                public EntityContainerEnum(List<T> entities) {
42      91                    this.entities = entities.AsReadOnly();
42      92                }
        93
54      94                public bool MoveNext() {
54      95                    position++;
54      96                    return position < entities.Count;
54      97                }
        98
 0      99                public void Reset() {
 0     100                    position = -1;
 0     101                }
       102
84     103                void IDisposable.Dispose() { }
       104
16     105                object IEnumerator.Current => Current;
       106
16     107                public T Current => entities[position];
       108            }
       109
       110            #endregion
       111
       112        }
       113    }
```

DIKUArcade.Entities.Shape

Summary

Class:	DIKUArcade.Entities.Shape
Assembly:	DIKUArcade
File(s):	/home/student/SU23Guest/DIKUGames/DIKUArcade/DIKUArcade/Entities/Shape.cs
Covered lines:	6
Uncovered lines:	55
Coverable lines:	61
Total lines:	114
Line coverage:	9.8% (6 of 61)
Covered branches:	1
Total branches:	4
Branch coverage:	25% (1 of 4)
Covered methods:	3
Total methods:	20
Method coverage:	15% (3 of 20)

Metrics

Method	Branch coverage	Cyclomatic complexity	Line coverage
get_Rotation()	100%	1	100%
get_Extent()	100%	1	100%
AsDynamicShape()	50.0%	2	100%
AsStationaryShape()	0%	2	0%
Scale(...)	100%	1	0%
Scale(...)	100%	1	0%
ScaleX(...)	100%	1	0%
ScaleY(...)	100%	1	0%
ScaleXFromCenter(...	100%	1	0%
ScaleYFromCenter(...	100%	1	0%
ScaleFromCenter(...)	100%	1	0%
ScaleFromCenter(...)	100%	1	0%
Move()	100%	1	0%
Move(...)	100%	1	0%
MoveX(...)	100%	1	0%
MoveY(...)	100%	1	0%

Move(...)	100%	1	0%
Rotate(...)	100%	1	0%
SetRotation(...)	100%	1	0%
SetPosition(...)	100%	1	0%

File(s)

/home/student/SU23Guest/DIKUGames/DIKUArcade/DIKUArcade/Entities/Shape.cs

#	Line	Line coverage
	1	`using DIKUArcade.Math;`
	2	
	3	`namespace DIKUArcade.Entities {`
	4	`    public class Shape {`
	5	`        /// <summary>`
	6	`        /// Shape's rotational angle measured in radians.`
	7	`        /// </summary>`
3	8	`        public float Rotation { get; set; }`
	9	
	10	`        /// <summary>`
	11	`        /// Basic Shape properties`
	12	`        /// </summary>`
	13	`        public Vec2F Position;`
21585	14	`        public Vec2F Extent { get; set; }`
	15	
	16	`        /// <summary>`
	17	`        /// Performs a downcast on this Shape instance to a`
	18	`        /// DynamicShape. If the downcast fails, a new`
	19	`        /// DynamicShape is returned instead with this Shape's`
	20	`        /// Position and Extent properties, and a default (0,0)`
	21	`        /// Direction vector.`
	22	`        /// </summary>`
	23	`        /// <returns></returns>`
84	24	`        public DynamicShape AsDynamicShape() {`
84	25	`            var shape = this as DynamicShape;`
84	26	`            return shape ?? new DynamicShape(Position, Extent);`
84	27	`        }`
	28	
	29	`        /// <summary>`

```
        /// Performs a downcast on this Shape instance to a
        /// StationaryShape. If the downcast fails, a new
        /// StationaryShape is returned instead with this Shape's
        /// Position and Extent properties.
        /// </summary>
        /// <returns></returns>
0       public StationaryShape AsStationaryShape() {
0           var sta = this as StationaryShape;
0           return sta ?? new StationaryShape(Position, Extent);
0       }

        // Do not reference other shapes if you intend to scale.
        // Use .Copy() or you might scale everything.
0       public void Scale(float scale) {
0           Extent *= scale;
0       }

0       public void Scale(Vec2F scalar) {
        // This is doing pairwise vector multiplication!
0           Extent *= scalar;
0       }

0       public void ScaleX(float scale) {
0           Extent.X *= scale;
0       }

0       public void ScaleY(float scale) {
0           Extent.Y *= scale;
0       }

0       public void ScaleXFromCenter(float scale) {
0           Position.X = (Position.X + Extent.X / 2.0f) - ((Extent.X / 2.0f) * scale);
0           Extent.X *= scale;
0       }

0       public void ScaleYFromCenter(float scale) {
0           Position.Y = (Position.Y + Extent.Y / 2.0f) - (Extent.Y / 2.0f * scale);
0           Extent.Y *= scale;
0       }

```

```
0    70            public void ScaleFromCenter(float scale) {
0    71                ScaleXFromCenter(scale);
0    72                ScaleYFromCenter(scale);
0    73            }
     74
0    75            public void ScaleFromCenter(Vec2F scalar) {
0    76                ScaleXFromCenter(scalar.X);
0    77                ScaleYFromCenter(scalar.Y);
0    78            }
     79
     80            /// <summary>
     81            /// Default Move method which does nothing.
     82            /// </summary>
0    83            public virtual void Move() {}
     84
0    85            public void Move(Vec2F mover) {
0    86                Position += mover;
0    87            }
     88
0    89            public void MoveX(float move) {
0    90                Position.X += move;
0    91            }
     92
0    93            public void MoveY(float move) {
0    94                Position.Y += move;
0    95            }
     96
0    97            public void Move(float x, float y) {
0    98                MoveX(x);
0    99                MoveY(y);
0   100            }
    101
0   102            public void Rotate(float angleRadians) {
0   103                Rotation += angleRadians;
0   104            }
    105
0   106            public void SetRotation(float angleRadians) {
0   107                Rotation = angleRadians;
0   108            }
    109
```

```
0    110            public void SetPosition(Vec2F newPosition) {
0    111                Position = newPosition;
0    112            }
     113        }
     114    }
```

DIKUArcade.Entities.StationaryShape

Summary

Class:	DIKUArcade.Entities.StationaryShape
Assembly:	DIKUArcade
File(s):	/student/SU23Guest/DIKUGames/DIKUArcade/DIKUArcade/Entities/StationaryShape.cs
Covered lines:	4
Uncovered lines:	7
Coverable lines:	11
Total lines:	23
Line coverage:	36.3% (4 of 11)
Covered branches:	0
Total branches:	0
Covered methods:	1
Total methods:	3
Method coverage:	33.3% (1 of 3)

Metrics

Method	Branch coverage	Cyclomatic complexity	Line coverage
.ctor(...)	100%	1	0%
.ctor(...)	100%	1	100%
op_Explicit(...)	100%	1	0%

File(s)

/student/SU23Guest/DIKUGames/DIKUArcade/DIKUArcade/Entities/StationaryShape.cs

#	Line	Line coverage
	1	`using DIKUArcade.Math;`
	2	
	3	`namespace DIKUArcade.Entities {`
	4	`    /// <summary>`
	5	`    /// Similar to DynamicShape, but does not contain direction information,`
	6	`    /// since a static object os not meant to be affected by game physics.`
	7	`    /// </summary>`
	8	`    public class StationaryShape : Shape {`

```
     0        public StationaryShape(float posX, float posY, float width, float height) {
     0            Position = new Vec2F(posX, posY);
     0            Extent = new Vec2F(width, height);
     0        }

34296        public StationaryShape(Vec2F pos, Vec2F extent) {
17148            Position = pos;
17148            Extent = extent;
17148        }

     0        public static explicit operator DynamicShape(StationaryShape sta) {
     0            return new DynamicShape(sta.Position, sta.Extent);
     0        }
         }
     }
```

DIKUArcade.Events.GameEventBus

Summary

Class:	DIKUArcade.Events.GameEventBus
Assembly:	DIKUArcade
File(s):	/home/student/SU23Guest/DIKUGames/DIKUArcade/DIKUArcade/Events/GameEventBus.cs
Covered lines:	86
Uncovered lines:	88
Coverable lines:	174
Total lines:	285
Line coverage:	49.4% (86 of 174)
Covered branches:	19
Total branches:	60
Branch coverage:	31.6% (19 of 60)
Covered methods:	11
Total methods:	19
Method coverage:	57.8% (11 of 19)

Metrics

Method	Branch coverage	Cyclomatic complexity	Line coverage
.ctor()	100%	1	100%
SwapTimedEventLists(	100%	1	100%
InitializeEventBus(.	83.33%	6	94.44%
Subscribe(...)	50.0%	4	60.0%
Unsubscribe(...)	0%	4	0%
RegisterTimedEvent(.	25.00%	4	57.14%
AddOrResetTimedEvent	0%	4	0%
CancelTimedEvent(...	0%	4	0%
ResetTimedEvent(...)	0%	2	0%
HasTimedEvent(...)	100%	1	0%
RegisterEvent(...)	50.0%	2	62.50%
ProcessTimedEvents()	75.00%	4	91.66%
ProcessEvents(...)	50.0%	2	84.00%
ProcessEventsSequent	0%	16	0%
ProcessEvents()	100%	2	100%
ProcessEventsSequent	0%	2	0%

BreakProcessing()	100%	1	100%
ResetBreakProcessing	100%	1	0%
Flush()	100%	4	100%

File(s)

/home/student/SU23Guest/DIKUGames/DIKUArcade/DIKUArcade/Events/GameEventBus.cs

#	Line	Line coverage
	1	`using System;`
	2	`using System.Collections.Generic;`
	3	`using System.Threading.Tasks;`
	4	`using DIKUArcade.Timers;`
	5	
	6	`namespace DIKUArcade.Events`
	7	`{`
	8	`    /// <summary>`
	9	`    /// Default implementation of GameEventBus (see below) which uses GameEventType`
	10	`    /// instead of a generic enum event type.`
	11	`    /// GameEventBus is the core module for processing events in the DIKUArcade game engine. Modules can register events`
	12	`    /// add them to the queues. Events are distinguished by event types to improve processing performance. Event process`
	13	`    /// can register/subscribe themself to receive events of a certain event type. For a single event, all processors ar`
	14	`    /// called with this event (broadcast semantic).`
	15	`    /// </summary>`
	16	`    public class GameEventBus : IGameEventBus, ITimedGameEventBus, IGameEventBusController`
	17	`    {`
1	18	`        private bool _initialized = false;`
	19	
	20	`        /// <summary>`
	21	`        /// Dictionary of registered event processors for a given game event type.`
	22	`        /// </summary>`
	23	`        private Dictionary<GameEventType, ICollection<IGameEventProcessor>> _eventProcessors;`
	24	`        /// <summary>`
	25	`        /// Dictionary of game event queues for different game event types.`
	26	`        /// </summary>`
	27	`        private Dictionary<GameEventType, GameEventQueue<GameEvent>> _eventQueues;`
	28	`        /// <summary>`
	29	`        /// Stops processing the pipeline, e.g. needed due real-time constraints.`
	30	`        /// </summary>`

```
1   31          private bool _breakExecution = false;
    32
    33          /// <summary>
    34          /// List of events which must be processed after a specified time interval has passed.
    35          /// We use a double-buffered system.
    36          /// </summary>
1   37          private int _activeTimedEventList = 0;
1   38          private int _inactiveTimedEventList = 1;
    39          private List<TimedGameEvent>[] _timedEventLists;
    40
41  41          private void SwapTimedEventLists() {
41  42              _activeTimedEventList = (_activeTimedEventList + 1) % 2;
41  43              _inactiveTimedEventList = (_inactiveTimedEventList + 1) % 2;
41  44          }
    45
    46
    47          /// <summary>
    48          /// Initialized the event bus to handle the specified event types.
    49          /// An exception is thrown if called on an already initialized GameEventBus.
    50          /// </summary>
    51          /// <exception cref="InvalidOperationException"></exception>
    52          public void InitializeEventBus(ICollection<GameEventType> eventTypeList)
1   53          {
1   54              if (_initialized) {
0   55                  throw new InvalidOperationException("GameEventBus is already initialized!");
    56              }
    57
1   58              _eventProcessors= new Dictionary<GameEventType, ICollection<IGameEventProcessor>>();
1   59              _eventQueues= new Dictionary<GameEventType, GameEventQueue<GameEvent>>();
    60
2   61              if (eventTypeList != null) {
13  62                  foreach (var eventType in eventTypeList)
5   63                  {
5   64                      _eventProcessors.Add(eventType, new List<IGameEventProcessor>());
5   65                      _eventQueues.Add(eventType, new GameEventQueue<GameEvent>());
5   66                  }
1   67              }
    68
1   69              _timedEventLists = new List<TimedGameEvent>[2] {
1   70                  new List<TimedGameEvent>(),
```

```
  1    71                    new List<TimedGameEvent>()
  1    72                };
       73
  1    74                _initialized = true;
  1    75            }
       76
       77            public void Subscribe(GameEventType eventType, IGameEventProcessor gameEventProcessor)
337    78            {
337    79                if (gameEventProcessor == default(IGameEventProcessor))
  0    80                    throw new ArgumentNullException("Parameter gameEventProcessor must not be null.");
       81
       82                try
337    83                {
337    84                    _eventProcessors?[eventType].Add(gameEventProcessor);
337    85                }
  0    86                catch (Exception e)
  0    87                {
  0    88                    throw new Exception($"Could not subscribe event processor. Check eventType! {e}");
       89                }
337    90            }
       91
       92            public void Unsubscribe(GameEventType eventType, IGameEventProcessor gameEventProcessor)
  0    93            {
  0    94                if (gameEventProcessor == default(IGameEventProcessor))
  0    95                    throw new ArgumentNullException("Parameter gameEventProcessor must not be null.");
       96
       97                try
  0    98                {
  0    99                    _eventProcessors?[eventType].Remove(gameEventProcessor);
  0   100                }
  0   101                catch (Exception e)
  0   102                {
  0   103                    throw new Exception($"Could not unsubsribe event processor. Check eventType or processor is unregistered
      104                }
  0   105            }
      106
      107
      108            #region TIMED_EVENTS
      109
      110            public void RegisterTimedEvent(GameEvent gameEvent, TimePeriod timePeriod)
```

```
6   111         {
    112             // do not insert already registered events:
6   113             if (gameEvent.Id != default(uint)) {
0   114                 if (_timedEventLists[_activeTimedEventList].Exists(e => e.GameEvent.Id == gameEvent.Id)) {
0   115                     return;
    116                 }
0   117             }
6   118             _timedEventLists[_activeTimedEventList].Add(new TimedGameEvent(timePeriod, gameEvent));
6   119         }
    120
0   121         public void AddOrResetTimedEvent(GameEvent gameEvent, TimePeriod timePeriod) {
0   122             if (gameEvent.Id != default(uint)) {
    123                 // search for an item which matches the Id of the specified event
0   124                 var search = _timedEventLists[_activeTimedEventList].FindIndex(e => e.GameEvent.Id == gameEvent.Id);
    125
0   126                 if (search >= 0) {
    127                     // event with Id already exists, so we reset its time period
0   128                     _timedEventLists[_activeTimedEventList][search] =
0   129                         new TimedGameEvent(timePeriod, _timedEventLists[_activeTimedEventList][search].GameEvent);
0   130                     return;
    131                 }
0   132             }
    133             // input event does not have an Id, or it has an Id but does not exist in list.
    134             // In either case, we add it.
0   135             _timedEventLists[_activeTimedEventList].Add(new TimedGameEvent(timePeriod, gameEvent));
0   136         }
    137
0   138         public bool CancelTimedEvent(uint eventId) {
0   139             bool cancelled = false;
0   140             _timedEventLists[_inactiveTimedEventList].Clear();
0   141             foreach (var e in _timedEventLists[_activeTimedEventList]) {
0   142                 if (e.GameEvent.Id != eventId) {
0   143                     _timedEventLists[_inactiveTimedEventList].Add(e);
0   144                 } else {
0   145                     cancelled = true;
0   146                 }
0   147             }
    148
    149             // swap the timed-event lists
0   150             SwapTimedEventLists();
```

```
0   151                 return cancelled;
0   152             }
    153
0   154             public bool ResetTimedEvent(uint eventId, TimePeriod timePeriod) {
0   155                 var search = _timedEventLists[_activeTimedEventList].FindIndex(e => e.GameEvent.Id == eventId);
0   156                 if (search >= 0) {
0   157                     _timedEventLists[_activeTimedEventList][search] =
0   158                         new TimedGameEvent(timePeriod, _timedEventLists[_activeTimedEventList][search].GameEvent);
0   159                     return true;
    160                 }
0   161                 return false;
0   162             }
    163
0   164             public bool HasTimedEvent(uint eventId) {
0   165                 return _timedEventLists[_activeTimedEventList].FindIndex(e => e.GameEvent.Id == eventId) >= 0;
0   166             }
    167
    168             #endregion // TIMED_EVENTS
    169
    170
    171             public void RegisterEvent(GameEvent gameEvent)
90  172             {
    173                 try
90  174                 {
90  175                     _eventQueues?[gameEvent.EventType].Enqueue(gameEvent);
90  176                 }
0   177                 catch (Exception e)
0   178                 {
0   179                     throw new Exception($"Could not register event. Did you Initialize the EventBus with {e.Message}");
    180                 }
90  181             }
    182
41  183             private void ProcessTimedEvents() {
41  184                 _timedEventLists[_inactiveTimedEventList].Clear();
    185
41  186                 var currentTime = Timers.StaticTimer.GetElapsedMilliseconds();
723 187                 foreach (var e in _timedEventLists[_activeTimedEventList]) {
200 188                     if (e.HasExpired(currentTime)) {
0   189                         RegisterEvent(e.GameEvent);
200 190                     } else {
```

```
200  191                        _timedEventLists[_inactiveTimedEventList].Add(e);
200  192                    }
200  193                }
 41  194                SwapTimedEventLists();
 41  195            }
     196
 41  197            public void ProcessEvents(IEnumerable<GameEventType> processOrder) {
 41  198                if(processOrder==default(IEnumerable<GameEventType>)) {
  0  199                    throw new ArgumentNullException();
     200                }
     201
 41  202                ProcessTimedEvents();
     203
 41  204                Parallel.ForEach<GameEventType>(processOrder, new Action<GameEventType, ParallelLoopState>(
194  205                    (eventType, loopState) => {
388  206                        if (_eventQueues != null) {
366  207                            while (!_eventQueues[eventType].IsEmpty()) {
 86  208                                var currentEvent = _eventQueues[eventType].Dequeue();
 86  209                                if (currentEvent.To != default(IGameEventProcessor))
  0  210                                {
  0  211                                    currentEvent.To.ProcessEvent(currentEvent);
  0  212                                }
 86  213                                else if (_eventProcessors != null)
 86  214                                {
6930  215                                    foreach (var eventProcessor in _eventProcessors[eventType]) {
2224  216                                        eventProcessor.ProcessEvent(currentEvent);
 41  217
4448  218                                        if (_breakExecution) loopState.Break();
2224  219                                    }
 86  220                                }
 86  221                            }
194  222                        }
235  223                }));
     224
     225                // semantic of Parallel.ForEach is it blocks until all parallel threads are finished
 41  226            }
     227
  0  228            public void ProcessEventsSequentially(IEnumerable<GameEventType> processOrder) {
  0  229                if(processOrder==default(IEnumerable<GameEventType>)) {
  0  230                    throw new ArgumentNullException();
```

```
        }

        ProcessTimedEvents();

        foreach(GameEventType eventType in processOrder) {
            if (_eventQueues != null) {
                while (!_eventQueues[eventType].IsEmpty()) {
                    var currentEvent = _eventQueues[eventType].Dequeue();
                    if (currentEvent.To != default(IGameEventProcessor))
                    {
                        currentEvent.To.ProcessEvent(currentEvent);
                    }
                    else if (_eventProcessors != null) {
                        foreach (var eventProcessor in _eventProcessors[eventType]) {
                            eventProcessor.ProcessEvent(currentEvent);
                            if (_breakExecution) return;
                        }
                    }
                }
            }
        }
    }

    public void ProcessEvents()
    {
        if (_eventQueues != null) ProcessEvents(_eventQueues.Keys);
    }

    public void ProcessEventsSequentially()
    {
        if (_eventQueues != null) ProcessEventsSequentially(_eventQueues.Keys);
    }

    public void BreakProcessing()
    {
        _breakExecution = true;
    }

    public void ResetBreakProcessing()
    {
```

```
0   271                 _breakExecution = false;
0   272             }
    273
    274             public void Flush()
1   275             {
1   276                 BreakProcessing();
    277
1   278                 if (_eventQueues == null) return;
13  279                 foreach (var eventType in _eventQueues.Keys)
5   280                 {
5   281                     _eventQueues[eventType].Flush();
5   282                 }
1   283             }
    284         }
    285     }
```

DIKUArcade.Events.GameEventQueue<T>

Summary

Class:	DIKUArcade.Events.GameEventQueue<T>
Assembly:	DIKUArcade
File(s):	ome/student/SU23Guest/DIKUGames/DIKUArcade/DIKUArcade/Events/GameEventQueue.cs
Covered lines:	17
Uncovered lines:	13
Coverable lines:	30
Total lines:	97
Line coverage:	56.6% (17 of 30)
Covered branches:	2
Total branches:	2
Branch coverage:	100% (2 of 2)
Covered methods:	5
Total methods:	12
Method coverage:	41.6% (5 of 12)

Metrics

Method	Branch coverage	Cyclomatic complexity	Line coverage
.ctor()	100%	1	100%
GetEnumerator()	100%	1	0%
System.Collections.I	100%	1	0%
CopyTo(...)	100%	1	0%
System.Collections.I	100%	1	0%
get_IsSynchronized()	100%	1	0%
get_SyncRoot()	100%	1	0%
System.Collections.G	100%	1	0%
Enqueue(...)	100%	1	100%
Dequeue()	100%	1	100%
IsEmpty()	100%	1	100%
Flush()	100%	2	100%

File(s)

ome/student/SU23Guest/DIKUGames/DIKUArcade/DIKUArcade/Events/GameEventQueue.cs

#	Line	Line coverage
	1	`using System;`
	2	`using System.Collections;`
	3	`using System.Collections.Concurrent;`
	4	`using System.Collections.Generic;`
	5	
	6	`namespace DIKUArcade.Events`
	7	`{`
	8	`    /// <summary>`
	9	`    /// Game event queue based on the concurrent queue implementation of the .NET framework`
	10	`    /// offering a simplified facade for the game event bus system.`
	11	`    /// </summary>`
	12	`    /// <typeparam name="TP">EventType data type.</typeparam>`
	13	`    public class GameEventQueue<TP> : ICollection, IReadOnlyCollection<TP>`
	14	`    {`
	15	`        /// <summary>`
	16	`        /// Core component of the event queue.`
	17	`        /// </summary>`
5	18	`        private readonly ConcurrentQueue<TP> _queue= new ConcurrentQueue<TP>();`
	19	
	20	`        /// <summary>`
	21	`        /// Enumerator access for event queue.`
	22	`        /// </summary>`
	23	`        /// <returns>IEnumerator of concurrent queue.</returns>`
	24	`        public IEnumerator<TP> GetEnumerator()`
0	25	`        {`
0	26	`            return ((IEnumerable<TP>)_queue).GetEnumerator();`
0	27	`        }`
	28	
	29	`        /// <summary>`
	30	`        /// Generic enumerator access for event queue.`
	31	`        /// </summary>`
	32	`        /// <returns>Generic IEnumerator of concurrent queue.</returns>`
	33	`        IEnumerator IEnumerable.GetEnumerator()`
0	34	`        {`
0	35	`            return GetEnumerator();`
0	36	`        }`
	37	
	38	`        /// <summary>`
	39	`        /// Copy semantics for fast array initialization and processing.`

```
        /// </summary>
        /// <param name="array">Copy queue elements to array.</param>
        /// <param name="index">Copy queue elements to which index position.</param>
        public void CopyTo(Array array, int index)
0       {
0           _queue.CopyTo((TP[])array, index);
0       }

0       int ICollection.Count => _queue.Count;

0       public bool IsSynchronized { get; }
0       public object SyncRoot { get; }

0       int IReadOnlyCollection<TP>.Count => _queue.Count;

        /// <summary>
        /// Enqueue a game event in the event queue.
        /// </summary>
        /// <param name="gameEvent">Event which is enqueued.</param>
        public void Enqueue(TP gameEvent)
90      {
90          _queue.Enqueue(gameEvent);
90      }

        /// <summary>
        /// Dequeues a game event from the event queue.
        /// </summary>
        /// <returns>A game event from event queue.</returns>
        public TP Dequeue()
86      {
            TP gameEvent;
86          _queue.TryDequeue(out gameEvent);
86          return gameEvent;
86      }

        /// <summary>
        /// Checks if the element queue is empty.
        /// </summary>
        /// <returns>true if game event queue is empty, otherwise false.</returns>
        public bool IsEmpty()
```

```
280    80            {
280    81                return _queue.IsEmpty;
280    82            }
       83
       84            /// <summary>
       85            /// Flushes all elements stored in the event queue.
       86            /// TODO: Method is slow and needs a rewrite.
       87            /// </summary>
       88            public void Flush()
  5    89            {
       90                TP gameEventDummy;
  9    91                while(!_queue.IsEmpty)
  4    92                {
  4    93                    _queue.TryDequeue(out gameEventDummy);
  4    94                }
  5    95            }
       96        }
       97    }
```

DIKUArcade.Events.Generic.GameEventBus<T>

Summary

Class:	DIKUArcade.Events.Generic.GameEventBus<T>
Assembly:	DIKUArcade
File(s):	dent/SU23Guest/DIKUGames/DIKUArcade/DIKUArcade/Events/Generic/GameEventBusT.cs
Covered lines:	0
Uncovered lines:	174
Coverable lines:	174
Total lines:	287
Line coverage:	0% (0 of 174)
Covered branches:	0
Total branches:	60
Branch coverage:	0% (0 of 60)
Covered methods:	0
Total methods:	19
Method coverage:	0% (0 of 19)

Metrics

Method	Branch coverage	Cyclomatic complexity	Line coverage
.ctor()	100%	1	0%
SwapTimedEventLists(	100%	1	0%
InitializeEventBus(.	0%	6	0%
Subscribe(...)	0%	4	0%
Unsubscribe(...)	0%	4	0%
RegisterTimedEvent(.	0%	4	0%
AddOrResetTimedEvent	0%	4	0%
CancelTimedEvent(...	0%	4	0%
ResetTimedEvent(...)	0%	2	0%
HasTimedEvent(...)	100%	1	0%
RegisterEvent(...)	0%	2	0%
ProcessTimedEvents()	0%	4	0%
ProcessEvents(...)	0%	2	0%
ProcessEventsSequent	0%	16	0%
ProcessEvents()	0%	2	0%
ProcessEventsSequent	0%	2	0%

BreakProcessing()	100%	1	0%
ResetBreakProcessing	100%	1	0%
Flush()	0%	4	0%

File(s)

dent/SU23Guest/DIKUGames/DIKUArcade/DIKUArcade/Events/Generic/GameEventBusT.cs

#	Line	Line coverage
	1	`using System;`
	2	`using System.Collections.Generic;`
	3	`using System.Threading.Tasks;`
	4	`using DIKUArcade.Timers;`
	5	
	6	`namespace DIKUArcade.Events.Generic`
	7	`{`
	8	`    /// <summary>`
	9	`    /// Generic version of the DIKUArcade.Events.GameEventBus class, which uses the generic type EventT`
	10	`    /// as the underlying event type enum.`
	11	`    /// GameEventBus is the core module for processing events in the DIKUArcade game engine. Modules can register events`
	12	`    /// add them to the queues. Events are distinguished by event types to improve processing performance. Event process`
	13	`    /// can register/subscribe themself to receive events of a certain event type. For a single event, all processors ar`
	14	`    /// called with this event (broadcast semantic).`
	15	`    /// </summary>`
	16	`    /// <typeparam name="EventT">Enumeration type representing type of game events.</typeparam>`
	17	`    public class GameEventBus<EventT> : IGameEventBus<EventT>, ITimedGameEventBus<EventT>,`
	18	`        IGameEventBusController<EventT> where EventT : System.Enum`
	19	`    {`
0	20	`        private bool _initialized = false;`
	21	
	22	`        /// <summary>`
	23	`        /// Dictionary of registered event processors for a given game event type.`
	24	`        /// </summary>`
	25	`        private Dictionary<EventT, ICollection<IGameEventProcessor<EventT>>> _eventProcessors;`
	26	`        /// <summary>`
	27	`        /// Dictionary of game event queues for different game event types.`
	28	`        /// </summary>`
	29	`        private Dictionary<EventT, GameEventQueue<GameEvent<EventT>>> _eventQueues;`
	30	`        /// <summary>`

```
        /// Stops processing the pipeline, e.g. needed due real-time constraints.
        /// </summary>
0       private bool _breakExecution = false;

        /// <summary>
        /// List of events which must be processed after a specified time interval has passed.
        /// We use a double-buffered system.
        /// </summary>
        private List<TimedGameEvent<EventT>>[] _timedEventLists;
0       private int _activeTimedEventList = 0;
0       private int _inactiveTimedEventList = 1;

0       private void SwapTimedEventLists() {
0           _activeTimedEventList = (_activeTimedEventList + 1) % 2;
0           _inactiveTimedEventList = (_inactiveTimedEventList + 1) % 2;
0       }


        /// <summary>
        /// Initialized the event bus to handle the specified event types.
        /// An exception is thrown if called on an already initialized GameEventBus.
        /// </summary>
        /// <exception cref="InvalidOperationException"></exception>
        public void InitializeEventBus(ICollection<EventT> eventTypeList)
0       {
0           if (_initialized) {
0               throw new InvalidOperationException("GameEventBus is already initialized!");
            }

0           _eventProcessors= new Dictionary<EventT, ICollection<IGameEventProcessor<EventT>>>();
0           _eventQueues= new Dictionary<EventT, GameEventQueue<GameEvent<EventT>>>();

0           if (eventTypeList != null) {
0               foreach (var eventType in eventTypeList)
0               {
0                   _eventProcessors.Add(eventType, new List<IGameEventProcessor<EventT>>());
0                   _eventQueues.Add(eventType, new GameEventQueue<GameEvent<EventT>>());
0               }
0           }

```

```
0    71                _timedEventLists = new List<TimedGameEvent<EventT>>[2] {
0    72                    new List<TimedGameEvent<EventT>>(),
0    73                    new List<TimedGameEvent<EventT>>()
0    74                };
     75
0    76                _initialized = true;
0    77            }
     78
     79            public void Subscribe(EventT eventType, IGameEventProcessor<EventT> gameEventProcessor)
0    80            {
0    81                if (gameEventProcessor == default(IGameEventProcessor<EventT>))
0    82                    throw new ArgumentNullException("Parameter gameEventProcessor must not be null.");
     83
     84                try
0    85                {
0    86                    _eventProcessors?[eventType].Add(gameEventProcessor);
0    87                }
0    88                catch (Exception e)
0    89                {
0    90                    throw new Exception($"Could not subscribe event processor. Check eventType! {e}");
     91                }
0    92            }
     93
     94            public void Unsubscribe(EventT eventType, IGameEventProcessor<EventT> gameEventProcessor)
0    95            {
0    96                if (gameEventProcessor == default(IGameEventProcessor<EventT>))
0    97                    throw new ArgumentNullException("Parameter gameEventProcessor must not be null.");
     98
     99                try
0   100                {
0   101                    _eventProcessors?[eventType].Remove(gameEventProcessor);
0   102                }
0   103                catch (Exception e)
0   104                {
0   105                    throw new Exception($"Could not unsubsribe event processor. Check eventType or processor is unregistered
    106                }
0   107            }
    108
    109
    110            #region TIMED_EVENTS
```

```
0   113         public void RegisterTimedEvent(GameEvent<EventT> gameEvent, TimePeriod timePeriod)
0   113         {
0   115             // do not insert already registered events:
0   115             if (gameEvent.Id != default(uint)) {
0   116                 if (_timedEventLists[_activeTimedEventList].Exists(e => e.GameEvent.Id == gameEvent.Id)) {
0   117                     return;
0   118                 }
0   119             }
0   120             _timedEventLists[_activeTimedEventList].Add(new TimedGameEvent<EventT>(timePeriod, gameEvent));
0   121         }
0   123         public void AddOrResetTimedEvent(GameEvent<EventT> gameEvent, TimePeriod timePeriod) {
0   124             if (gameEvent.Id != default(uint)) {
0   126                 var search = _timedEventLists[_activeTimedEventList].FindIndex(e => e.GameEvent.Id == gameEvent.Id);
0   128                 if (search >= 0) {
0   130                     _timedEventLists[_activeTimedEventList][search] =
0   131                         new TimedGameEvent<EventT>(timePeriod, _timedEventLists[_activeTimedEventList][search].GameEvent
0   132                     return;
0   134             }
0   137             _timedEventLists[_activeTimedEventList].Add(new TimedGameEvent<EventT>(timePeriod, gameEvent));
0   138         }
0   140         public bool CancelTimedEvent(uint eventId) {
0   141             bool cancelled = false;
0   142             _timedEventLists[_inactiveTimedEventList].Clear();
0   143             foreach (var e in _timedEventLists[_activeTimedEventList]) {
0   144                 if (e.GameEvent.Id != eventId) {
0   145                     _timedEventLists[_inactiveTimedEventList].Add(e);
0   146                 } else {
0   147                     cancelled = true;
0   148                 }
0   149             }
```

```
0 151                 // swap the timed-event lists
0 152                 SwapTimedEventLists();
0 153                 return cancelled;
0 154             }
  155
0 156             public bool ResetTimedEvent(uint eventId, TimePeriod timePeriod) {
0 157                 var search = _timedEventLists[_activeTimedEventList].FindIndex(e => e.GameEvent.Id == eventId);
0 158                 if (search >= 0) {
0 159                     _timedEventLists[_activeTimedEventList][search] =
0 160                         new TimedGameEvent<EventT>(timePeriod, _timedEventLists[_activeTimedEventList][search].GameEvent);
0 161                     return true;
  162                 }
0 163                 return false;
0 164             }
  165
0 166             public bool HasTimedEvent(uint eventId) {
0 167                 return _timedEventLists[_activeTimedEventList].FindIndex(e => e.GameEvent.Id == eventId) >= 0;
0 168             }
  169
  170             #endregion // TIMED_EVENTS
  171
  172
  173             public void RegisterEvent(GameEvent<EventT> gameEvent)
0 174             {
  175                 try
0 176                 {
0 177                     _eventQueues?[gameEvent.EventType].Enqueue(gameEvent);
0 178                 }
0 179                 catch (Exception e)
0 180                 {
0 181                     throw new Exception($"Could not register event. Did you Initialize the EventBus with {e.Message}");
  182                 }
0 183             }
  184
0 185             private void ProcessTimedEvents() {
0 186                 _timedEventLists[_inactiveTimedEventList].Clear();
  187
0 188                 var currentTime = Timers.StaticTimer.GetElapsedMilliseconds();
0 189                 foreach (var e in _timedEventLists[_activeTimedEventList]) {
0 190                     if (e.HasExpired(currentTime)) {
```

```
0   191                         RegisterEvent(e.GameEvent);
0   192                     } else {
0   193                         _timedEventLists[_inactiveTimedEventList].Add(e);
0   194                     }
0   195                 }
0   196                 SwapTimedEventLists();
0   197             }
    198
0   199             public void ProcessEvents(IEnumerable<EventT> processOrder) {
0   200                 if(processOrder==default(IEnumerable<EventT>)) {
0   201                     throw new ArgumentNullException();
    202                 }
    203
0   204                 ProcessTimedEvents();
    205
0   206                 Parallel.ForEach<EventT>(processOrder, new Action<EventT, ParallelLoopState>(
0   207                     (eventType, loopState) => {
0   208                         if (_eventQueues != null) {
0   209                             while (!_eventQueues[eventType].IsEmpty()) {
0   210                                 var currentEvent = _eventQueues[eventType].Dequeue();
0   211                                 if (currentEvent.To != default(IGameEventProcessor<EventT>))
0   212                                 {
0   213                                     currentEvent.To.ProcessEvent(currentEvent);
0   214                                 }
0   215                                 else if (_eventProcessors != null)
0   216                                 {
0   217                                     foreach (var eventProcessor in _eventProcessors[eventType]) {
0   218                                         eventProcessor.ProcessEvent(currentEvent);
0   219
0   220                                         if (_breakExecution) loopState.Break();
0   221                                     }
0   222                                 }
0   223                             }
0   224                         }
0   225                 }));
    226
    227                 // semantic of Parallel.ForEach is it blocks until all parallel threads are finished
0   228             }
    229
0   230             public void ProcessEventsSequentially(IEnumerable<EventT> processOrder) {
```

```
0   231                 if(processOrder==default(IEnumerable<EventT>)) {
0   232                     throw new ArgumentNullException();
    233                 }
    234
0   235                 ProcessTimedEvents();
    236
0   237                 foreach(EventT eventType in processOrder) {
0   238                     if (_eventQueues != null) {
0   239                         while (!_eventQueues[eventType].IsEmpty()) {
0   240                             var currentEvent = _eventQueues[eventType].Dequeue();
0   241                             if (currentEvent.To != default(IGameEventProcessor<EventT>))
0   242                             {
0   243                                 currentEvent.To.ProcessEvent(currentEvent);
0   244                             }
0   245                             else if (_eventProcessors != null) {
0   246                                 foreach (var eventProcessor in _eventProcessors[eventType]) {
0   247                                     eventProcessor.ProcessEvent(currentEvent);
0   248                                     if (_breakExecution) return;
0   249                                 }
0   250                             }
0   251                         }
0   252                     }
0   253                 }
0   254             }
    255
    256             public void ProcessEvents()
0   257             {
0   258                 if (_eventQueues != null) ProcessEvents(_eventQueues.Keys);
0   259             }
    260
    261             public void ProcessEventsSequentially()
0   262             {
0   263                 if (_eventQueues != null) ProcessEventsSequentially(_eventQueues.Keys);
0   264             }
    265
    266             public void BreakProcessing()
0   267             {
0   268                 _breakExecution = true;
0   269             }
    270
```

```
        271            public void ResetBreakProcessing()
0       272            {
0       273                _breakExecution = false;
0       274            }
        275
        276            public void Flush()
0       277            {
0       278                BreakProcessing();
        279
0       280                if (_eventQueues == null) return;
0       281                foreach (var eventType in _eventQueues.Keys)
0       282                {
0       283                    _eventQueues[eventType].Flush();
0       284                }
0       285            }
        286        }
        287    }
```

DIKUArcade.Events.Generic.TimedGameEvent<T>

Summary

Class:	DIKUArcade.Events.Generic.TimedGameEvent<T>
Assembly:	DIKUArcade
File(s):	nt/SU23Guest/DIKUGames/DIKUArcade/DIKUArcade/Events/Generic/TimedGameEventT.cs
Covered lines:	0
Uncovered lines:	13
Coverable lines:	13
Total lines:	46
Line coverage:	0% (0 of 13)
Covered branches:	0
Total branches:	0
Covered methods:	0
Total methods:	4
Method coverage:	0% (0 of 4)

Metrics

Method	Branch coverage	Cyclomatic complexity	Line coverage
get_GameEvent()	100%	1	0%
.ctor(...)	100%	1	0%
HasExpired()	100%	1	0%
HasExpired(...)	100%	1	0%

File(s)

nt/SU23Guest/DIKUGames/DIKUArcade/DIKUArcade/Events/Generic/TimedGameEventT.cs

#	Line	Line coverage
	1	`using DIKUArcade.Timers;`
	2	
	3	`namespace DIKUArcade.Events.Generic`
	4	`{`
	5	`    /// <summary>`
	6	`    /// Generic version of the DIKUArcade.Events.TimedGameEvent struct.`
	7	`    /// Represents a GameEvent together with an expiration time.`

```
        /// When a TimedGameEvent has expired it is ready for processing by a GameEventBus.
        /// </summary>
        /// <typeparam name="EventT">Enumeration type representing type of game events.</typeparam>
        public struct TimedGameEvent<EventT> where EventT : System.Enum
        {
            /// <summary>
            /// The GameEvent<EventT> which this object wraps around.
            /// </summary>
0           public GameEvent<EventT> GameEvent { get; private set; }

            private readonly TimePeriod timeSpan;
            private readonly long timeOfCreation;

0           public TimedGameEvent(TimePeriod timeSpan, GameEvent<EventT> gameEvent) {
0               this.timeSpan = timeSpan;
0               GameEvent = gameEvent;

0               timeOfCreation = StaticTimer.GetElapsedMilliseconds();
0           }

            /// <summary>
            /// Measure time and check if the event is ready for processing.
            /// </summary>
0           public bool HasExpired() {
0               var now = StaticTimer.GetElapsedMilliseconds();
0               return (now - timeOfCreation) > timeSpan.ToMilliseconds();
0           }

            /// <summary>
            /// Measure time and check if the event is ready for processing,
            /// but where current timestamp is provided in milliseconds.
            /// This is useful if checking multiple TimedEvents in sequence without
            /// having to get current timestamp for each one.
            /// </summary>
0           public bool HasExpired(long currentTime) {
0               return (currentTime - timeOfCreation) > timeSpan.ToMilliseconds();
0           }
        }
    }
```

DIKUArcade.Events.TimedGameEvent

Summary

Class:	DIKUArcade.Events.TimedGameEvent
Assembly:	DIKUArcade
File(s):	ome/student/SU23Guest/DIKUGames/DIKUArcade/DIKUArcade/Events/TimedGameEvent.cs
Covered lines:	9
Uncovered lines:	4
Coverable lines:	13
Total lines:	43
Line coverage:	69.2% (9 of 13)
Covered branches:	0
Total branches:	0
Covered methods:	3
Total methods:	4
Method coverage:	75% (3 of 4)

Metrics

Method	Branch coverage	Cyclomatic complexity	Line coverage
get_GameEvent()	100%	1	100%
.ctor(...)	100%	1	100%
HasExpired()	100%	1	0%
HasExpired(...)	100%	1	100%

File(s)

ome/student/SU23Guest/DIKUGames/DIKUArcade/DIKUArcade/Events/TimedGameEvent.cs

#	Line	Line coverage
	1	`using DIKUArcade.Timers;`
	2	
	3	`namespace DIKUArcade.Events {`
	4	`    /// <summary>`
	5	`    /// Represents a GameEvent together with an expiration time.`
	6	`    /// When a TimedGameEvent has expired it is ready for processing by a GameEventBus.`
	7	`    /// </summary>`

```
        8        public struct TimedGameEvent
        9        {
       10            /// <summary>
       11            /// The GameEvent which this object wraps around.
       12            /// </summary>
   6   13            public GameEvent GameEvent { get; private set; }
       14
       15            private readonly TimePeriod timeSpan;
       16            private readonly long timeOfCreation;
       17
   6   18            public TimedGameEvent(TimePeriod timeSpan, GameEvent gameEvent) {
   6   19                this.timeSpan = timeSpan;
   6   20                GameEvent = gameEvent;
       21
   6   22                timeOfCreation = StaticTimer.GetElapsedMilliseconds();
   6   23            }
       24
       25            /// <summary>
       26            /// Measure time and check if the event is ready for processing.
       27            /// </summary>
   0   28            public bool HasExpired() {
   0   29                var now = StaticTimer.GetElapsedMilliseconds();
   0   30                return (now - timeOfCreation) > timeSpan.ToMilliseconds();
   0   31            }
       32
       33            /// <summary>
       34            /// Measure time and check if the event is ready for processing,
       35            /// but where current timestamp is provided in milliseconds.
       36            /// This is useful if checking multiple TimedEvents in sequence without
       37            /// having to get current timestamp for each one.
       38            /// </summary>
 200   39            public bool HasExpired(long currentTimeMs) {
 200   40                return (currentTimeMs - timeOfCreation) > timeSpan.ToMilliseconds();
 200   41            }
       42        }
       43    }
```

DIKUArcade.Graphics.Animation

Summary

Class:	DIKUArcade.Graphics.Animation
Assembly:	DIKUArcade
File(s):	/home/student/SU23Guest/DIKUGames/DIKUArcade/DIKUArcade/Graphics/Animation.cs
Covered lines:	0
Uncovered lines:	15
Coverable lines:	15
Total lines:	44
Line coverage:	0% (0 of 15)
Covered branches:	0
Total branches:	0
Covered methods:	0
Total methods:	7
Method coverage:	0% (0 of 7)

Metrics

Method	Branch coverage	Cyclomatic complexity	Line coverage
get_Duration()	100%	1	0%
get_Shape()	100%	1	0%
get_Stride()	100%	1	0%
.ctor()	100%	1	0%
IsActive()	100%	1	0%
RenderAnimation()	100%	1	0%
ResetAnimation()	100%	1	0%

File(s)

/home/student/SU23Guest/DIKUGames/DIKUArcade/DIKUArcade/Graphics/Animation.cs

#	Line	Line coverage
	1	`using System;`
	2	`using DIKUArcade.Entities;`
	3	`using DIKUArcade.Timers;`
	4	

```
namespace DIKUArcade.Graphics {
    public class Animation {
        /// <summary>
        /// Duration of this animation, in milliseconds
        /// </summary>
        public int Duration { get; set; }

        /// <summary>
        /// Position and Extent of this animation
        /// </summary>
        public StationaryShape Shape { get; set; }

        /// <summary>
        /// ImageStride used for animation
        /// </summary>
        public ImageStride Stride { get; set; }

        private double timeOfCreation;

        public Animation() {
            timeOfCreation = StaticTimer.GetElapsedMilliseconds();
        }

        /// <summary>
        /// The animation is still considered active if the specified duration
        /// in milliseconds has not yet passed.
        /// </summary>
        public bool IsActive() {
            return timeOfCreation + Duration > StaticTimer.GetElapsedMilliseconds();
        }

        public void RenderAnimation() {
            Stride.Render(Shape);
        }

        public void ResetAnimation() {
            timeOfCreation = StaticTimer.GetElapsedMilliseconds();
        }
    }
}
```

DIKUArcade.Graphics.AnimationContainer

Summary

Class:	DIKUArcade.Graphics.AnimationContainer
Assembly:	DIKUArcade
File(s):	udent/SU23Guest/DIKUGames/DIKUArcade/DIKUArcade/Graphics/AnimationContainer.cs
Covered lines:	0
Uncovered lines:	40
Coverable lines:	40
Total lines:	72
Line coverage:	0% (0 of 40)
Covered branches:	0
Total branches:	14
Branch coverage:	0% (0 of 14)
Covered methods:	0
Total methods:	5
Method coverage:	0% (0 of 5)

Metrics

Method	Branch coverage	Cyclomatic complexity	Line coverage
get_Occupied()	100%	1	0%
.ctor(...)	0%	2	0%
ResetContainer()	0%	2	0%
AddAnimation(...)	0%	4	0%
RenderAnimations()	0%	6	0%

File(s)

udent/SU23Guest/DIKUGames/DIKUArcade/DIKUArcade/Graphics/AnimationContainer.cs

#	Line	Line coverage
	1	`using System;`
	2	`using DIKUArcade.Entities;`
	3	
	4	`namespace DIKUArcade.Graphics {`
	5	`    public class AnimationContainer {`

```
        internal class OccupyValue<T> {
0           public bool Occupied { get; set; }
            public T Value;
        }

        private OccupyValue<Animation>[] container;
        private int size;

0       public AnimationContainer(int size) {
0           if (size < 0) {
0               throw new ArgumentOutOfRangeException(
0                   $"Cannot instantiate Animation container with negative size: {size}");
            }

0           container = new OccupyValue<Animation>[size];
0           this.size = size;
0           ResetContainer();
0       }

        /// <summary>
        /// Clear this container of all bound animation objects
        /// </summary>
0       public void ResetContainer() {
0           for (int i = 0; i < size; i++) {
0               container[i] = new OccupyValue<Animation>()
0                   {Occupied = false, Value = new Animation() {
0                       Duration = 0, Shape = new StationaryShape(0.0f, 0.0f, 0.0f, 0.0f)
0                   }};
0           }
0       }

        /// <summary>
        /// Add an animation to this container. Return true if successful, otherwise false.
        /// </summary>
        /// <param name="shape"></param>
        /// <param name="duration"></param>
        /// <param name="stride"></param>
0       public bool AddAnimation(Shape shape, int duration, ImageStride stride) {
0           for (int i = 0; i < size; i++) {
0               var anim = container[i];
```

```
0    46                    if (!anim.Occupied) {
0    47                        anim.Occupied = true;
0    48                        anim.Value.Shape.Position = shape.Position;
0    49                        anim.Value.Shape.Extent = shape.Extent;
0    50                        anim.Value.Duration = duration;
0    51                        anim.Value.Stride = stride;
0    52                        anim.Value.ResetAnimation();
0    53                        return true;
     54                    }
0    55                }
0    56                return false;
0    57            }
     58
     59            /// <summary>
     60            /// Render all animation objects currently bound to this container
     61            /// </summary>
0    62            public void RenderAnimations() {
0    63                foreach (var animation in container) {
0    64                    if (animation.Occupied && animation.Value.IsActive()) {
0    65                        animation.Value.RenderAnimation();
0    66                    } else {
0    67                        animation.Occupied = false;
0    68                    }
0    69                }
0    70            }
     71        }
     72    }
```

DIKUArcade.Graphics.Camera

Summary

Class:	DIKUArcade.Graphics.Camera
Assembly:	DIKUArcade
File(s):	/home/student/SU23Guest/DIKUGames/DIKUArcade/DIKUArcade/Graphics/Camera.cs
Covered lines:	0
Uncovered lines:	12
Coverable lines:	12
Total lines:	26
Line coverage:	0% (0 of 12)
Covered branches:	0
Total branches:	0
Covered methods:	0
Total methods:	3
Method coverage:	0% (0 of 3)

Metrics

Method	Branch coverage	Cyclomatic complexity	Line coverage
ScaleBy(...)	100%	1	0%
OffsetBy(...)	100%	1	0%
setZoom()	100%	1	0%

File(s)

/home/student/SU23Guest/DIKUGames/DIKUArcade/DIKUArcade/Graphics/Camera.cs

#	Line	Line coverage
	1	`using DIKUArcade.Math;`
	2	`using OpenTK.Graphics.OpenGL;`
	3	
	4	`namespace DIKUArcade.Graphics {`
	5	
	6	`public abstract class Camera {`
	7	`public Vec2F Offset;`
	8	`public float Scale;`

```
        9
0      10            public void ScaleBy(float scalar) {
0      11                Scale *= scalar;
0      12                setZoom();
0      13            }
       14
0      15            public void OffsetBy(Vec2F offsetBy) {
0      16                Offset += offsetBy;
0      17            }
       18
0      19            private void setZoom() {
0      20                GL.MatrixMode(MatrixMode.Projection);
0      21                GL.LoadIdentity();
       22                //GL.Ortho(-1.0, 1.0, -1.0, 1.0, 0.0, 4.0);
0      23                GL.Ortho(0.0, 1.0 * Scale,0.0,1.0 * Scale, 0.0, 4.0);
0      24            }
       25        }
       26    }
```

DIKUArcade.Graphics.ChaseCamera

Summary

Class:	DIKUArcade.Graphics.ChaseCamera
Assembly:	DIKUArcade
File(s):	home/student/SU23Guest/DIKUGames/DIKUArcade/DIKUArcade/Graphics/ChaseCamera.cs
Covered lines:	0
Uncovered lines:	19
Coverable lines:	19
Total lines:	50
Line coverage:	0% (0 of 19)
Covered branches:	0
Total branches:	10
Branch coverage:	0% (0 of 10)
Covered methods:	0
Total methods:	2
Method coverage:	0% (0 of 2)

Metrics

Method	Branch coverage	Cyclomatic complexity	Line coverage
.ctor(...)	0%	2	0%
EnqueueDirection(...	0%	8	0%

File(s)

home/student/SU23Guest/DIKUGames/DIKUArcade/DIKUArcade/Graphics/ChaseCamera.cs

#	Line	Line coverage
	1	`using DIKUArcade.Entities;`
	2	`using DIKUArcade.Math;`
	3	
	4	`using System.Collections.Generic;`
	5	`namespace DIKUArcade.Graphics {`
	6	
	7	`/// <summary>A camera that takes a direction </summary>`
	8	`    public class ChaseCamera : Camera {`

```
        public Shape WorldShape;

        public DynamicShape cameraShape;
        // We want to expose the camera position, as it is quite nice to know what we are looking at.
        //public Vec2F CameraPos() { return cameraShape.Position; }

0       private Vec2F baseOffset = new Vec2F(0.5f, 0.5f);


        private Queue<Vec2F> directionQueue;
        // Set the frame delay of the camera
        private const int CAMERA_DELAY = 20;


0       public ChaseCamera(StationaryShape worldShape) {

0           cameraShape = new DynamicShape(0.5f, 0.5f, 0.0f, 0.0f);
0           Offset = baseOffset - cameraShape.Position;//new Vec2F(0f, 0f);
0           Scale = 1f;

0           WorldShape = worldShape;
             // Initialize the queue and fill it with 0-vectors to make the camera lag behind by CAMERA_DELAY seconds
0           directionQueue = new Queue<Vec2F>(CAMERA_DELAY);
0           for (int i = 0; i < CAMERA_DELAY; i++) { directionQueue.Enqueue(new Vec2F(0f,0f)); }
0       }

0       public void EnqueueDirection(Vec2F direction) {
0           cameraShape.Direction = directionQueue.Dequeue();
0           directionQueue.Enqueue(direction);

0           cameraShape.Move();

            // Update camera offset and clamp it to the worldshape
0           Offset = baseOffset - cameraShape.Position;
0           if (-Offset.X      < WorldShape.Position.X)                       { Offset.X = -WorldShape.Position.X; }
0           if (-Offset.X + 1f > WorldShape.Position.X + WorldShape.Extent.X) { Offset.X = -(WorldShape.Position.X + Wor
0           if (-Offset.Y      < WorldShape.Position.Y)                       { Offset.Y = -WorldShape.Position.Y; }
0           if (-Offset.Y + 1f > WorldShape.Position.Y + WorldShape.Extent.Y) { Offset.Y = -(WorldShape.Position.Y + Wor

0       }
```

```
        }
    }
```

DIKUArcade.Graphics.DynamicCamera

Summary

Class:	DIKUArcade.Graphics.DynamicCamera
Assembly:	DIKUArcade
File(s):	me/student/SU23Guest/DIKUGames/DIKUArcade/DIKUArcade/Graphics/DynamicCamera.cs
Covered lines:	0
Uncovered lines:	46
Coverable lines:	46
Total lines:	79
Line coverage:	0% (0 of 46)
Covered branches:	0
Total branches:	20
Branch coverage:	0% (0 of 20)
Covered methods:	0
Total methods:	3
Method coverage:	0% (0 of 3)

Metrics

Method	Branch coverage	Cyclomatic complexity	Line coverage
.ctor(...)	100%	1	0%
OffsetRelativeTo(...	0%	20	0%
Render()	100%	1	0%

File(s)

me/student/SU23Guest/DIKUGames/DIKUArcade/DIKUArcade/Graphics/DynamicCamera.cs

#	Line	Line coverage
	1	`using System.IO;`
	2	`using DIKUArcade.Entities;`
	3	`using DIKUArcade.Math;`
	4	`namespace DIKUArcade.Graphics {`
	5	
	6	`    public class DynamicCamera : Camera {`
	7	`        public Shape WorldShape;`

```
        8
        9        private DynamicShape innerBounds;
       10        private Entity overlay;
       11
       12        private Vec2F displacement;
0      13        public DynamicCamera(StationaryShape worldShape) {
0      14            Offset = new Vec2F(0.0f, 0.0f);
0      15            Scale = 1f;
0      16            WorldShape = worldShape;
0      17            innerBounds = new DynamicShape(0.25f, 0.25f, 0.5f, 0.5f);
       18            //innerBounds.ScaleFromCenter(0.5f);
0      19            displacement = innerBounds.Position;
0      20            overlay = new Entity(innerBounds, new Image(Path.Combine("Assets", "Images", "Overlay.png")));
       21
0      22            System.Console.WriteLine("Offset is: {0}", Offset);
0      23            System.Console.WriteLine("inner is: {0}", innerBounds.Position);
0      24        }
       25
0      26        public void OffsetRelativeTo(Vec2F offsetRelativeTo) {
       27            // Check if "safe" inside the inner bounds and exit early
       28            // The magic constant 0.03f is the players width and height
0      29            if (innerBounds.Position.X <= offsetRelativeTo.X
0      30                && offsetRelativeTo.X <= innerBounds.Position.X + innerBounds.Extent.X - 0.03f
0      31                && innerBounds.Position.Y <= offsetRelativeTo.Y
0      32                && offsetRelativeTo.Y <= innerBounds.Position.Y + innerBounds.Extent.Y - 0.03f) {
       33                    //innerBounds.Direction = new Vec2F(0.0f, 0.0f);
0      34                    return;
       35                }
0      36            else {
       37                // Calculate new offset
       38                // If the player has driven out the left side of the box
0      39                if (offsetRelativeTo.X < innerBounds.Position.X)
0      40                {
0      41                    innerBounds.Position.X = offsetRelativeTo.X;
       42                    //innerBounds.Direction.X = offsetRelativeTo.X - innerBounds.Position.X;
0      43                }
0      44                else if (offsetRelativeTo.X > innerBounds.Position.X + innerBounds.Extent.X - 0.03f) {
0      45                    innerBounds.Position.X = (offsetRelativeTo.X - innerBounds.Extent.X - 0.03f);
       46                    //innerBounds.Direction.X = (offsetRelativeTo.X - (innerBounds.Position.X + innerBounds.Extent.X -0.
0      47                }
```

```
                    //innerBounds.Direction *= 3.0f;
                    //innerBounds.Move();

                    Offset = displacement - innerBounds.Position;
                    // Then check if outside the world and move stuff back
                    // Stop at the edge of the world
                    if (-Offset.X < WorldShape.Position.X) { // Left side
                            Offset.X = -WorldShape.Position.X;
                            innerBounds.Position.X = offsetRelativeTo.X;//WorldShape.Position.X + innerBounds.Extent.X / 2.0
                        }
                    if (-Offset.X + 1f > WorldShape.Position.X + WorldShape.Extent.X) { // Right side
                         Offset.X = -(WorldShape.Position.X + WorldShape.Extent.X - 1f);
                         innerBounds.Position.X = offsetRelativeTo.X - 0.03f;//WorldShape.Extent.X - innerBounds.Extent.X;
                        }
                    if (-Offset.Y      < WorldShape.Position.Y) // Top
                        { Offset.Y = -WorldShape.Position.Y;
                        innerBounds.Position.Y = offsetRelativeTo.Y; }
                    if (-Offset.Y + 1f > WorldShape.Position.Y + WorldShape.Extent.Y) // Bottom
                        {
                            Offset.Y = -(WorldShape.Position.Y + WorldShape.Extent.Y - 1.0f);
                            innerBounds.Position.Y = offsetRelativeTo.Y - 0.03f;
                        }

                }

            }

            public void Render() {
                overlay.Image.Render(overlay.Shape, this);
            }
        }
    }
```

DIKUArcade.Graphics.FollowCamera

Summary

Class:	DIKUArcade.Graphics.FollowCamera
Assembly:	DIKUArcade
File(s):	ome/student/SU23Guest/DIKUGames/DIKUArcade/DIKUArcade/Graphics/FollowCamera.cs
Covered lines:	0
Uncovered lines:	13
Coverable lines:	13
Total lines:	23
Line coverage:	0% (0 of 13)
Covered branches:	0
Total branches:	8
Branch coverage:	0% (0 of 8)
Covered methods:	0
Total methods:	2
Method coverage:	0% (0 of 2)

Metrics

Method	Branch coverage	Cyclomatic complexity	Line coverage
.ctor(...)	100%	1	0%
OffsetRelativeTo(...	0%	8	0%

File(s)

ome/student/SU23Guest/DIKUGames/DIKUArcade/DIKUArcade/Graphics/FollowCamera.cs

#	Line	Line coverage
	1	`using DIKUArcade.Entities;`
	2	`using DIKUArcade.Math;`
	3	`namespace DIKUArcade.Graphics {`
	4	
	5	`public class FollowCamera : Camera {`
	6	`public Shape WorldShape;`
	7	
0	8	`private Vec2F baseOffset = new Vec2F(0.5f, 0.5f);`

```
0     9            public FollowCamera(StationaryShape worldShape) {
0    10                Offset = new Vec2F(0f, 0f);
0    11                Scale = 1f;
0    12                WorldShape = worldShape;
0    13            }
     14
0    15            public void OffsetRelativeTo(Vec2F offsetRelativeTo) {
0    16                Offset = baseOffset - offsetRelativeTo;
0    17                if (-Offset.X      < WorldShape.Position.X)                          { Offset.X = -WorldShape.Position.X;
0    18                if (-Offset.X + 1f > WorldShape.Position.X + WorldShape.Extent.X) { Offset.X = -(WorldShape.Position.X + Wor
0    19                if (-Offset.Y      < WorldShape.Position.Y)                          { Offset.Y = -WorldShape.Position.Y;
0    20                if (-Offset.Y + 1f > WorldShape.Position.Y + WorldShape.Extent.Y) { Offset.Y = -(WorldShape.Position.Y + Wor
0    21            }
     22        }
     23    }
```

DIKUArcade.Graphics.Image

Summary

Class:	DIKUArcade.Graphics.Image
Assembly:	DIKUArcade
File(s):	/home/student/SU23Guest/DIKUGames/DIKUArcade/DIKUArcade/Graphics/Image.cs
Covered lines:	3
Uncovered lines:	12
Coverable lines:	15
Total lines:	28
Line coverage:	20% (3 of 15)
Covered branches:	0
Total branches:	0
Covered methods:	1
Total methods:	5
Method coverage:	20% (1 of 5)

Metrics

Method	Branch coverage	Cyclomatic complexity	Line coverage
.ctor(...)	100%	1	100%
.ctor(...)	100%	1	0%
Render(...)	100%	1	0%
Render(...)	100%	1	0%
GetTexture()	100%	1	0%

File(s)

/home/student/SU23Guest/DIKUGames/DIKUArcade/DIKUArcade/Graphics/Image.cs

#	Line	Line coverage
	1	`using System;`
	2	`using DIKUArcade.Entities;`
	3	
	4	`namespace DIKUArcade.Graphics {`
	5	`    public class Image : IBaseImage {`
	6	

```
            private Texture texture;

10502       public Image(string imageFile) {
 5251           texture = new Texture(imageFile);
 5251       }

    0       public Image(Texture texture) {
    0           this.texture = texture;
    0       }

    0       public void Render(Shape shape) {
    0           texture.Render(shape);
    0       }
    0       public void Render(Shape shape, Camera camera) {
    0           texture.Render(shape, camera);
    0       }

    0       public Texture GetTexture() {
    0           return texture;
    0       }
        }
    }
```

DIKUArcade.Graphics.ImageStride

Summary

Class:	DIKUArcade.Graphics.ImageStride
Assembly:	DIKUArcade
File(s):	home/student/SU23Guest/DIKUGames/DIKUArcade/DIKUArcade/Graphics/ImageStride.cs
Covered lines:	0
Uncovered lines:	87
Coverable lines:	87
Total lines:	191
Line coverage:	0% (0 of 87)
Covered branches:	0
Total branches:	32
Branch coverage:	0% (0 of 32)
Covered methods:	0
Total methods:	10
Method coverage:	0% (0 of 10)

Metrics

Method	Branch coverage	Cyclomatic complexity	Line coverage
.ctor(...)	0%	6	0%
.ctor(...)	0%	6	0%
.ctor(...)	0%	6	0%
CreateStrides(...)	0%	2	0%
StartAnimation()	100%	1	0%
StopAnimation()	100%	1	0%
SetAnimationFrequenc	0%	2	0%
ChangeAnimationFrequ	0%	2	0%
Render(...)	0%	8	0%
Render(...)	100%	1	0%

File(s)

home/student/SU23Guest/DIKUGames/DIKUArcade/DIKUArcade/Graphics/ImageStride.cs

#	Line	Line coverage

```
1   using System;
2   using System.Collections.Generic;
3   using System.IO;
4   using DIKUArcade.Timers;
5   using DIKUArcade.Entities;
6   using DIKUArcade.Math;
7   using DIKUArcade.Utilities;
8
9   namespace DIKUArcade.Graphics {
10      /// <summary>
11      /// Image stride to show animations based on a list of textures
12      /// and an animation frequency.
13      /// </summary>
14      public class ImageStride : IBaseImage {
15          private int animFrequency;
16
17          private double lastTime;
18          private bool animate;
19
20          private List<Texture> textures;
21          private readonly int maxImageCount;
22          private int currentImageCount;
23
24          /// <summary>
25          /// This value is only added for random animation offset,
26          /// e.g. 100 objects created at the same time with the same
27          /// animation frequency will not change texture at the exact
28          /// same time.
29          /// </summary>
30          private double timerOffset;
31
32          /// <summary>
33          ///
34          /// </summary>
35          /// <param name="milliseconds">Time between consecutive frames</param>
36          /// <param name="imageFiles">List of image files to include in strides</param>
37          public ImageStride(int milliseconds, params string[] imageFiles) {
38              if (milliseconds < 0) {
39                  throw new ArgumentException("milliseconds must be a positive integer");
40              }
```

```
0    41                animFrequency = milliseconds;
0    42                animate = true;
     43
0    44                int imgs = imageFiles.Length;
0    45                if (imgs == 0) {
     46                    // ReSharper disable once NotResolvedInText
0    47                    throw new ArgumentNullException("At least one image file must be specified");
     48                }
0    49                maxImageCount = imgs - 1;
0    50                currentImageCount = RandomGenerator.Generator.Next(imgs);
0    51                timerOffset = RandomGenerator.Generator.Next(100);
     52
0    53                textures = new List<Texture>(imgs);
0    54                foreach (string imgFile in imageFiles)
0    55                {
0    56                    textures.Add(new Texture(imgFile));
0    57                }
0    58            }
     59
     60            /// <summary>
     61            ///
     62            /// </summary>
     63            /// <param name="milliseconds">Time between consecutive frames</param>
     64            /// <param name="images">List of images to include in strides</param>
0    65            public ImageStride(int milliseconds, params Image[] images) {
0    66                if (milliseconds < 0) {
0    67                    throw new ArgumentException("milliseconds must be a positive integer");
     68                }
0    69                animFrequency = milliseconds;
0    70                animate = true;
     71
0    72                int imgs = images.Length;
0    73                if (imgs == 0) {
     74                    // ReSharper disable once NotResolvedInText
0    75                    throw new ArgumentNullException("at least one image file must be specified");
     76                }
     77
0    78                maxImageCount = imgs - 1;
0    79                currentImageCount = RandomGenerator.Generator.Next(imgs);
0    80                timerOffset = RandomGenerator.Generator.Next(100);
```

```
0   82                textures = new List<Texture>(imgs);
0   83                foreach (Image img in images)
0   84                {
0   85                    textures.Add(img.GetTexture());
0   86                }
0   87            }
    88
0   89            public ImageStride(int milliseconds, List<Image> images) {
0   90                if (milliseconds < 0) {
0   91                    throw new ArgumentException("milliseconds must be a positive integer");
    92                }
0   93                animFrequency = milliseconds;
0   94                animate = true;
    95
0   96                int imgs = images.Count;
0   97                if (imgs == 0) {
    98                    // ReSharper disable once NotResolvedInText
0   99                    throw new ArgumentNullException("at least one image file must be specified");
    100               }
    101
0   102               maxImageCount = imgs - 1;
0   103               currentImageCount = RandomGenerator.Generator.Next(imgs);
0   104               timerOffset = RandomGenerator.Generator.Next(100);
    105
0   106               textures = new List<Texture>(imgs);
0   107               foreach (Image img in images)
0   108               {
0   109                   textures.Add(img.GetTexture());
0   110               }
0   111           }
    112
    113           /// <summary>
    114           /// Create a List of images from an image stride file.
    115           /// </summary>
    116           /// <param name="numStrides">Total number of strides in the image</param>
    117           /// <param name="imagePath">The relative path to the image</param>
    118           /// <returns>A list of Image objects, each corresponding to a stride of the image.</returns>
0   119           public static List<Image> CreateStrides(int numStrides, string imagePath) {
0   120               var res = new List<Image>();
```

```
0   121
0   122             for (int i = 0; i < numStrides; i++) {
0   123                 res.Add(new Image(new Texture(imagePath, i, numStrides)));
0   124             }
0   125             return res;
0   126         }
    127
    128         /// <summary>
    129         /// Restart animation for this ImageStride object
    130         /// </summary>
0   131         public void StartAnimation() {
0   132             animate = true;
0   133             lastTime = StaticTimer.GetElapsedMilliseconds();
0   134         }
    135
    136         /// <summary>
    137         /// Halt animation for this ImageStride object
    138         /// </summary>
0   139         public void StopAnimation() {
0   140             animate = false;
0   141         }
    142
    143         /// <summary>
    144         /// Change the animation frequency for this ImageStride object
    145         /// </summary>
    146         /// <param name="milliseconds"></param>
    147         /// <exception cref="ArgumentException">milliseconds must be a positive integer</exception>
0   148         public void SetAnimationFrequency(int milliseconds) {
0   149             if (milliseconds < 0) {
0   150                 throw new ArgumentException("milliseconds must be a positive integer");
    151             }
0   152             animFrequency = milliseconds;
0   153         }
    154
    155         /// <summary>
    156         /// Relatively change the animation frequency for this ImageStride object
    157         /// </summary>
    158         /// <param name="millisecondsChange"></param>
    159         /// <exception cref="ArgumentException">milliseconds must be a positive integer</exception>
0   160         public void ChangeAnimationFrequency(int millisecondsChange) {
```

```
0   161                 animFrequency += millisecondsChange;
0   162                 if (animFrequency < 0) {
0   163                     animFrequency = 0;
0   164                 }
0   165             }
    166
    167             /// <summary>
    168             /// Render this ImageStride object onto the currently active drawing window
    169             /// </summary>
    170             /// <param name="shape">The Shape object for the rendered image</param>
0   171             public void Render(Shape shape) {
    172                 // measure elapsed time
0   173                 double elapsed = StaticTimer.GetElapsedMilliseconds() + timerOffset;
    174
    175                 // the desired number of milliseconds has passed, change texture stride
0   176                 if (animFrequency > 0 && animate && elapsed - lastTime > animFrequency) {
0   177                     lastTime = elapsed;
    178
0   179                     currentImageCount =
0   180                         (currentImageCount >= maxImageCount) ? 0 : currentImageCount + 1;
0   181                 }
    182
    183                 // render the current texture object
0   184                 textures[currentImageCount].Render(shape);
0   185             }
0   186             public void Render(Shape shape, Camera camera) {
0   187                 throw new NotImplementedException();
    188
    189             }
    190         }
    191     }
```

DIKUArcade.Graphics.NoImage

Summary

Class:	DIKUArcade.Graphics.NoImage
Assembly:	DIKUArcade
File(s):	/home/student/SU23Guest/DIKUGames/DIKUArcade/DIKUArcade/Graphics/NoImage.cs
Covered lines:	0
Uncovered lines:	3
Coverable lines:	3
Total lines:	11
Line coverage:	0% (0 of 3)
Covered branches:	0
Total branches:	0
Covered methods:	0
Total methods:	3
Method coverage:	0% (0 of 3)

Metrics

Method	Branch coverage	Cyclomatic complexity	Line coverage
.ctor()	100%	1	0%
Render(...)	100%	1	0%
Render(...)	100%	1	0%

File(s)

/home/student/SU23Guest/DIKUGames/DIKUArcade/DIKUArcade/Graphics/NoImage.cs

#	Line	Line coverage
	1	`using System;`
	2	`using DIKUArcade.Entities;`
	3	
	4	`namespace DIKUArcade.Graphics {`
	5	`    /// A stub for an image, to use with entities that are non-drawable`
	6	`    public class NoImage : IBaseImage {`
0	7	`        public NoImage() {}`
0	8	`        public void Render(Shape shape) {}`

```
0     9            public void Render(Shape shape, Camera camera) {}
     10        }
     11    }
```

DIKUArcade.Graphics.StaticCamera

Summary

Class:	DIKUArcade.Graphics.StaticCamera
Assembly:	DIKUArcade
File(s):	ome/student/SU23Guest/DIKUGames/DIKUArcade/DIKUArcade/Graphics/StaticCamera.cs
Covered lines:	0
Uncovered lines:	4
Coverable lines:	4
Total lines:	14
Line coverage:	0% (0 of 4)
Covered branches:	0
Total branches:	0
Covered methods:	0
Total methods:	1
Method coverage:	0% (0 of 1)

Metrics

Method	Branch coverage	Cyclomatic complexity	Line coverage
.ctor()	100%	1	0%

File(s)

ome/student/SU23Guest/DIKUGames/DIKUArcade/DIKUArcade/Graphics/StaticCamera.cs

#	Line	Line coverage
	1	`using DIKUArcade.Math;`
	2	`namespace DIKUArcade.Graphics {`
	3	
	4	`    public class StaticCamera : Camera {`
0	5	`        public StaticCamera() {`
0	6	`            Offset = new Vec2F(0f, 0f);`
0	7	`            Scale = 1f;`
0	8	`        }`
	9	
	10	

```


        }
    }
```

DIKUArcade.Graphics.Text

Summary

Class:	DIKUArcade.Graphics.Text
Assembly:	DIKUArcade
File(s):	/home/student/SU23Guest/DIKUGames/DIKUArcade/DIKUArcade/Graphics/Text.cs
Covered lines:	79
Uncovered lines:	62
Coverable lines:	141
Total lines:	300
Line coverage:	56% (79 of 141)
Covered branches:	6
Total branches:	78
Branch coverage:	7.6% (6 of 78)
Covered methods:	8
Total methods:	17
Method coverage:	47% (8 of 17)

Metrics

Method	Branch coverage	Cyclomatic complexity	Line coverage
.ctor(...)	100%	1	100%
CreateBitmapTexture(	100%	1	100%
BindTexture()	100%	1	100%
UnbindTexture()	100%	1	100%
GetShape()	100%	1	0%
SetText(...)	100%	1	100%
SetFontSize(...)	0%	2	0%
SetFont(...)	0%	4	0%
SetColor(...)	0%	12	0%
SetColor(...)	50.0%	12	87.50%
SetColor(...)	0%	16	0%
SetColor(...)	0%	16	0%
SetColor(...)	0%	16	0%
SetColor(...)	100%	1	0%
CreateMatrix()	100%	1	100%
ScaleText(...)	100%	1	0%

RenderText()	100%	1	100%

File(s)

/home/student/SU23Guest/DIKUGames/DIKUArcade/DIKUArcade/Graphics/Text.cs

#	Line	Line coverage

```
1    using System;
2    using System.Drawing;
3    using System.Drawing.Imaging;
4    using System.Drawing.Text;
5    using OpenTK.Graphics.OpenGL;
6    using DIKUArcade.Entities;
7    using DIKUArcade.Math;
8    using OpenTK.Mathematics;
9
10   namespace DIKUArcade.Graphics {
11       public class Text {
12           // TODO: Add method for centering text (vertically, horizontally) within its shape!
13           /// <summary>
14           /// OpenGL texture handle
15           /// </summary>
16           private int textureId;
17
18           /// <summary>
19           /// The string value for the text
20           /// </summary>
21           private string text;
22
23           /// <summary>
24           /// The font size for the text string
25           /// </summary>
26           private int fontSize;
27
28           /// <summary>
29           /// The position and size of the text
30           /// </summary>
31           private StationaryShape shape;
32
```

```
        33          /// <summary>
        34          /// The color for the text
        35          /// </summary>
        36          private System.Drawing.Color color;
        37
        38          /// <summary>
        39          /// The font family of the text.
        40          /// </summary>
        41          private System.Drawing.Font font;
        42
2650    43          public Text(string text, Vec2F pos, Vec2F extent) {
1325    44              this.text = text;
1325    45              shape = new StationaryShape(pos, extent);
1325    46              color = System.Drawing.Color.Black;
1325    47              fontSize = 50;
1325    48              font = new Font("Arial", fontSize);
        49
        50              // create a texture id
1325    51              textureId = GL.GenTexture();
        52
        53              // bind this new texture id
1325    54              BindTexture();
        55
        56              // set texture properties, filters, blending functions, etc.
1325    57              GL.TexParameter(TextureTarget.Texture2D, TextureParameterName.TextureMagFilter, (int)All.Linear);
1325    58              GL.TexParameter(TextureTarget.Texture2D, TextureParameterName.TextureMinFilter, (int)All.Linear);
        59
1325    60              GL.Enable(EnableCap.Blend);
1325    61              GL.BlendFunc(BlendingFactor.SrcAlpha, BlendingFactor.OneMinusSrcAlpha);
        62
1325    63              GL.Enable(EnableCap.DepthTest);
1325    64              GL.DepthFunc(DepthFunction.Lequal);
        65
1325    66              GL.Enable(EnableCap.Texture2D);
1325    67              GL.Enable(EnableCap.AlphaTest);
        68
1325    69              GL.AlphaFunc(AlphaFunction.Gequal, 0.5f);
        70
        71              // unbind this new texture
1325    72              UnbindTexture();
```

```
        73
        74                // create a texture
1325    75                CreateBitmapTexture();
1325    76            }
        77
        78            // This method assumes that
2800    79            private void CreateBitmapTexture() {
2800    80                BindTexture();
        81
2800    82                System.Drawing.Bitmap textBmp = new System.Drawing.Bitmap(500, 500); // match window size
        83
        84                // just allocate memory, so we can update efficiently using TexSubImage2D
2800    85                GL.TexImage2D(TextureTarget.Texture2D, 0, PixelInternalFormat.Rgba, textBmp.Width, textBmp.Height, 0,
2800    86                    OpenTK.Graphics.OpenGL.PixelFormat.Bgra, PixelType.UnsignedByte, IntPtr.Zero);
        87
2800    88                using (System.Drawing.Graphics gfx = System.Drawing.Graphics.FromImage(textBmp))
2800    89                {
2800    90                    gfx.Clear(System.Drawing.Color.Transparent);
        91                    // TODO: Could create an enumeration for choosing btw different font families!
2800    92                    Font drawFont = font;
2800    93                    SolidBrush drawBrush = new SolidBrush(color);
        94
        95                    // TODO: Maybe we should not use shape.Position here, because different coordinate system !!?
2800    96                    System.Drawing.PointF drawPoint = new System.Drawing.PointF(shape.Position.X, shape.Position.Y);
        97
2800    98                    gfx.DrawString(text, drawFont, drawBrush, drawPoint); // Draw as many strings as you need
2800    99                }
       100
2800   101                BitmapData data = textBmp.LockBits(new System.Drawing.Rectangle(0, 0, textBmp.Width, textBmp.Height),
2800   102                    ImageLockMode.ReadOnly, System.Drawing.Imaging.PixelFormat.Format32bppArgb);
2800   103                GL.TexImage2D(TextureTarget.Texture2D, 0, PixelInternalFormat.Rgba, textBmp.Width, textBmp.Height, 0,
2800   104                    OpenTK.Graphics.OpenGL.PixelFormat.Bgra, PixelType.UnsignedByte, data.Scan0);
2800   105                textBmp.UnlockBits(data);
       106
2800   107                UnbindTexture();
2800   108            }
       109
4128   110            private void BindTexture() {
4128   111                GL.BindTexture(TextureTarget.Texture2D, textureId);
4128   112            }
```

```
        113
4128    114         private void UnbindTexture() {
4128    115             GL.BindTexture(TextureTarget.Texture2D, 0); // 0 is invalid texture id
4128    116         }
        117
   0    118         public StationaryShape GetShape() {
   0    119             return shape;
   0    120         }
        121
        122         #region ChangeTextProperties
        123
        124         /// <summary>
        125         /// Set the text string for this Text object.
        126         /// </summary>
        127         /// <param name="newText">The new text string</param>
 120    128         public void SetText(string newText) {
 120    129             text = newText;
 120    130             CreateBitmapTexture();
 120    131         }
        132
        133         /// <summary>
        134         /// Set the font size for this Text object.
        135         /// </summary>
        136         /// <param name="newSize">The new font size</param>
        137         /// <exception cref="ArgumentOutOfRangeException">Font size must be a
        138         /// positive integer.</exception>
   0    139         public void SetFontSize(int newSize) {
   0    140             if (newSize < 0) {
        141                 // ReSharper disable once NotResolvedInText
   0    142                 throw  new ArgumentOutOfRangeException("Font size must be a positive integer");
        143             }
   0    144             fontSize = newSize;
   0    145             CreateBitmapTexture();
   0    146         }
        147
        148         /// <summary>
        149         /// Set the font for this Text object, if the font is installed.
        150         /// If the font is not installed defaults to Arial.
        151         /// </summary>
        152         /// <param name="fontfamily">The name of the font family</param>
```

```
    0    153            public void SetFont(string fontfamily) {
         154                // The loop below checks if said font is installed, if not defaults to Arial.
    0    155                var fontsCollection = new InstalledFontCollection();
    0    156                foreach (var fontFamily in fontsCollection.Families) {
    0    157                    if (fontFamily.Name == fontfamily) {
    0    158                        font = new Font(fontfamily, fontSize);
    0    159                        break;
         160                    }
    0    161                }
         162
    0    163                CreateBitmapTexture();
    0    164            }
         165
         166            /// <summary>
         167            /// Change text color
         168            /// </summary>
         169            /// <param name="vec">Vec3F containing the RGB color values.</param>
         170            /// <exception cref="ArgumentOutOfRangeException">Normalized color values must be
         171            /// between 0 and 1.</exception>
    0    172            public void SetColor(Vec3F vec) {
    0    173                if (vec.X < 0.0f || vec.X > 1.0f ||
    0    174                    vec.Y < 0.0f || vec.Y > 1.0f ||
    0    175                    vec.Z < 0.0f || vec.Z > 1.0f) {
    0    176                    throw new ArgumentOutOfRangeException($"RGB Color values must be between 0 and 1: {vec}");
         177                }
    0    178                color = System.Drawing.Color.FromArgb((int)(vec.X * 255.0f), (int)(vec.Y * 255.0f), (int)(vec.Z * 255.0f));
    0    179                CreateBitmapTexture();
    0    180            }
         181
         182            /// <summary>
         183            /// Change text color
         184            /// </summary>
         185            /// <param name="vec">Vec3I containing the RGB color values.</param>
         186            /// <exception cref="ArgumentOutOfRangeException">Color values must be
         187            /// between 0 and 255.</exception>
 1355    188            public void SetColor(Vec3I vec) {
 1355    189                if (vec.X < 0 || vec.X > 255 ||
 1355    190                    vec.Y < 0 || vec.Y > 255 ||
 1355    191                    vec.Z < 0 || vec.Z > 255) {
    0    192                    throw new ArgumentOutOfRangeException($"RGB Color values must be between 0 and 255: {vec}");
```

```
            }
            color = System.Drawing.Color.FromArgb(vec.X, vec.Y, vec.Z);
            CreateBitmapTexture();
        }

        /// <summary>
        /// Change text color
        /// </summary>
        /// <param name="vec">Vec4I containing the ARGB color values.</param>
        /// <exception cref="ArgumentOutOfRangeException">Color values must be
        /// between 0 and 255.</exception>
        public void SetColor(int a, int r, int g, int b) {
            if (a < 0 || a > 255 ||
                r < 0 || r > 255 ||
                g < 0 || g > 255 ||
                b < 0 || b > 255) {
                throw new ArgumentOutOfRangeException($"ARGB Color values must be between 0 and 255: {a} {r} {g} {b}");
            }
            color = System.Drawing.Color.FromArgb(a, r, g, b);
            CreateBitmapTexture();
        }

        /// <summary>
        /// Change text color
        /// </summary>
        /// <param name="vec">Vec4I containing the ARGB color values.</param>
        /// <exception cref="ArgumentOutOfRangeException">Color values must be
        /// between 0 and 255.</exception>
        public void SetColor(Vec4I vec) {
            if (vec.X < 0 || vec.X > 255 ||
                vec.Y < 0 || vec.Y > 255 ||
                vec.Z < 0 || vec.Z > 255 ||
                vec.W < 0 || vec.W > 255) {
                throw new ArgumentOutOfRangeException($"ARGB Color values must be between 0 and 255: {vec}");
            }
            color = System.Drawing.Color.FromArgb(vec.X, vec.Y, vec.Z, vec.W);
            CreateBitmapTexture();
        }

        /// <summary>
```

```
        /// Change text color
        /// </summary>
        /// <param name="vec">Vec3F containing the RGB color values.</param>
        /// <exception cref="ArgumentOutOfRangeException">Normalized color values must be
        /// between 0 and 1.</exception>
0       public void SetColor(Vec4F vec) {
0           if (vec.X < 0.0f || vec.X > 1.0f ||
0               vec.Y < 0.0f || vec.Y > 1.0f ||
0               vec.Z < 0.0f || vec.Z > 1.0f ||
0               vec.W < 0.0f || vec.W > 1.0f) {
0               throw new ArgumentOutOfRangeException($"ARGB Color values must be between 0 and 1: {vec}");
            }
0           color = System.Drawing.Color.FromArgb((int)(vec.X * 255.0f), (int)(vec.Y * 255.0f), (int)(vec.Z * 255.0f), (
0           CreateBitmapTexture();
0       }

        /// <summary>
        /// Change text color
        /// </summary>
        /// <param name="newColor">System.Drawing.Color containing new color channel values.</param>
0       public void SetColor(System.Drawing.Color newColor) {
0           color = newColor;
0           CreateBitmapTexture();
0       }

        #endregion

3       private Matrix4 CreateMatrix() {
            // ensure that rotation is performed around the center of the shape
            // instead of the bottom-left corner
3           var halfX = shape.Extent.X / 2.0f;
3           var halfY = shape.Extent.Y / 2.0f;

3           return Matrix4.CreateTranslation(-halfX, -halfY, 0.0f) *
3                  Matrix4.CreateRotationZ(shape.Rotation) *
3                  Matrix4.CreateTranslation(shape.Position.X + halfX, shape.Position.Y + halfY,
3                      0.0f);
3       }

0       public void ScaleText(float scale) {
```

```
0   273                 shape.Position *= scale;
0   274                 shape.Scale(scale);
0   275             }
    276
3   277             public void RenderText() {
    278                 // bind this texture
3   279                 BindTexture();
    280
    281                 // render this texture
3   282                 Matrix4 modelViewMatrix = CreateMatrix();
3   283                 GL.MatrixMode(MatrixMode.Modelview);
3   284                 GL.LoadMatrix(ref modelViewMatrix);
    285
3   286                 GL.Color4(1f,1f,1f,1f);
3   287                 GL.Begin(PrimitiveType.Quads);
    288
6   289                 GL.TexCoord2(0, 1); GL.Vertex2(0.0f, 0.0f);                      // Top Left
6   290                 GL.TexCoord2(0, 0); GL.Vertex2(0.0f, shape.Extent.Y);            // Bottom Left
6   291                 GL.TexCoord2(1, 0); GL.Vertex2(shape.Extent.X, shape.Extent.Y);  // Bottom Right
6   292                 GL.TexCoord2(1, 1); GL.Vertex2(shape.Extent.X, 0.0f);            // Top Right
    293
3   294                 GL.End();
    295
    296                 // unbind this texture
3   297                 UnbindTexture();
3   298             }
    299         }
    300     }
```

DIKUArcade.Graphics.Texture

Summary

Class:	DIKUArcade.Graphics.Texture
Assembly:	DIKUArcade
File(s):	/home/student/SU23Guest/DIKUGames/DIKUArcade/DIKUArcade/Graphics/Texture.cs
Covered lines:	38
Uncovered lines:	92
Coverable lines:	130
Total lines:	226
Line coverage:	29.2% (38 of 130)
Covered branches:	3
Total branches:	14
Branch coverage:	21.4% (3 of 14)
Covered methods:	3
Total methods:	8
Method coverage:	37.5% (3 of 8)

Metrics

Method	Branch coverage	Cyclomatic complexity	Line coverage
.ctor(...)	75.00%	4	94.11%
.ctor(...)	0%	10	0%
BindTexture()	100%	1	100%
UnbindTexture()	100%	1	100%
CreateMatrix(...)	100%	1	0%
CreateMatrix(...)	100%	1	0%
Render(...)	100%	1	0%
Render(...)	100%	1	0%

File(s)

/home/student/SU23Guest/DIKUGames/DIKUArcade/DIKUArcade/Graphics/Texture.cs

#	Line	Line coverage
	1	`using System;`
	2	`using System.Drawing.Imaging;`

```
                using System.IO;
                using OpenTK.Mathematics;
                using OpenTK.Graphics.OpenGL;
                using DIKUArcade.Entities;

                namespace DIKUArcade.Graphics {
                    public class Texture {
                        /// <summary>
                        /// OpenGL texture handle
                        /// </summary>
                        public static double offsetX = 0.0;
                        public static double offsetY = 0.0;
                        private int textureId;

10502           public Texture(string filename) {
                            // create a texture id
5251                        textureId = GL.GenTexture();

                            // bind this new texture id
5251                        BindTexture();

                            // find base path
5251                        var dir = new DirectoryInfo(Path.GetDirectoryName(
5251                            System.Reflection.Assembly.GetExecutingAssembly().Location));

15753                       while (dir.Name != "bin")
10502                       {
10502                           dir = dir.Parent;
10502                       }
5251                        dir = dir.Parent;

                            // load image file
5251                        var path = Path.Combine(dir.FullName.ToString(), filename);
5251                        if (!File.Exists(path))
0                           {
0                               throw new FileNotFoundException($"Error: The file \"{path}\" does not exist.");
                            }
5251                        System.Drawing.Bitmap image = new System.Drawing.Bitmap(path);
5251                        BitmapData data = image.LockBits(new System.Drawing.Rectangle(0, 0, image.Width, image.Height),
5251                            ImageLockMode.ReadOnly, System.Drawing.Imaging.PixelFormat.Format32bppPArgb);
```

```
        43
        44                // attach it to OpenGL context
5251    45                GL.TexImage2D(TextureTarget.Texture2D, 0, PixelInternalFormat.Rgba,
5251    46                    data.Width, data.Height, 0, OpenTK.Graphics.OpenGL.PixelFormat.Bgra,
5251    47                    PixelType.UnsignedByte, data.Scan0);
        48
5251    49                image.UnlockBits(data);
        50
        51                // set texture properties, filters, blending functions, etc.
5251    52                GL.TexParameter(TextureTarget.Texture2D, TextureParameterName.TextureMinFilter,
5251    53                    (int)TextureMinFilter.Linear);
5251    54                GL.TexParameter(TextureTarget.Texture2D, TextureParameterName.TextureMagFilter,
5251    55                    (int)TextureMagFilter.Linear);
        56
5251    57                GL.Enable(EnableCap.Blend);
5251    58                GL.BlendFunc(BlendingFactor.SrcAlpha, BlendingFactor.OneMinusSrcAlpha);
5251    59                GL.Enable(EnableCap.DepthTest);
5251    60                GL.DepthFunc(DepthFunction.Lequal);
        61
5251    62                GL.Enable(EnableCap.Texture2D);
5251    63                GL.Enable(EnableCap.AlphaTest);
        64
5251    65                GL.AlphaFunc(AlphaFunction.Gequal, 0.5f);
        66
        67                // unbind the texture
5251    68                UnbindTexture();
5251    69            }
        70
   0    71            public Texture(string filename, int currentStride, int stridesInImage)
   0    72            {
   0    73                if (currentStride < 0 || currentStride >= stridesInImage || stridesInImage < 0)
   0    74                {
   0    75                    throw new ArgumentOutOfRangeException(
   0    76                        $"Invalid stride numbers: ({currentStride}/{stridesInImage})");
        77                }
        78                // create a texture id
   0    79                textureId = GL.GenTexture();
        80
        81                // bind this new texture id
   0    82                BindTexture();
```

```
0   85          var dir = new DirectoryInfo(Path.GetDirectoryName(
0   86              System.Reflection.Assembly.GetExecutingAssembly().Location));

0   88          while (dir.Name != "bin")
0   89          {
0   90              dir = dir.Parent;
0   91          }
0   92          dir = dir.Parent;

0   95          var path = Path.Combine(dir.FullName.ToString(), filename);
0   96          if (!File.Exists(path))
0   97          {
0   98              throw new FileNotFoundException($"Error: The file \"{path}\" does not exist.");
0  100          System.Drawing.Bitmap image = new System.Drawing.Bitmap(path);
0  101          var width = (int)((float)image.Width / (float)stridesInImage);
0  102          var posX = currentStride * width;
0  103          BitmapData data = image.LockBits(new System.Drawing.Rectangle(posX, 0, width, image.Height),
0  104              ImageLockMode.ReadOnly, System.Drawing.Imaging.PixelFormat.Format32bppPArgb);

0  107          GL.TexImage2D(TextureTarget.Texture2D, 0, PixelInternalFormat.Rgba,
0  108              data.Width, data.Height, 0, OpenTK.Graphics.OpenGL.PixelFormat.Bgra,
0  109              PixelType.UnsignedByte, data.Scan0);

0  111          image.UnlockBits(data);

0  114          GL.TexParameter(TextureTarget.Texture2D, TextureParameterName.TextureMinFilter,
0  115              (int)TextureMinFilter.Linear);
0  116          GL.TexParameter(TextureTarget.Texture2D, TextureParameterName.TextureMagFilter,
0  117              (int)TextureMagFilter.Linear);

0  119          GL.Enable(EnableCap.Blend);
0  120          GL.BlendFunc(BlendingFactor.SrcAlpha, BlendingFactor.OneMinusSrcAlpha);
0  121          GL.Enable(EnableCap.DepthTest);
0  122          GL.DepthFunc(DepthFunction.Lequal);
```

```
        123
    0   124                 GL.Enable(EnableCap.Texture2D);
    0   125                 GL.Enable(EnableCap.AlphaTest);
        126
    0   127                 GL.AlphaFunc(AlphaFunction.Gequal, 0.5f);
        128
        129                 // unbind the texture
    0   130                 UnbindTexture();
    0   131             }
        132
        133             private void BindTexture()
 5251   134             {
 5251   135                 GL.BindTexture(TextureTarget.Texture2D, textureId);
 5251   136             }
        137
        138             private void UnbindTexture()
 5251   139             {
 5251   140                 GL.BindTexture(TextureTarget.Texture2D, 0); // 0 is invalid texture id
 5251   141             }
        142
        143             private Matrix4 CreateMatrix(Shape shape)
    0   144             {
        145                 // ensure that rotation is performed around the center of the shape
        146                 // instead of the bottom-left corner
    0   147                 var halfX = shape.Extent.X / 2.0f;
    0   148                 var halfY = shape.Extent.Y / 2.0f;
        149
    0   150                 return Matrix4.CreateTranslation(-halfX, -halfY, 0.0f) *
    0   151                        Matrix4.CreateRotationZ(shape.Rotation) *
    0   152                        Matrix4.CreateTranslation(shape.Position.X + halfX, shape.Position.Y + halfY,
    0   153                            0.0f);
    0   154             }
        155
        156             // Render things that are affected by a camera (if the game has one)
        157             private Matrix4 CreateMatrix(Shape shape, Camera camera)
    0   158             {
        159                 // ensure that rotation is performed around the center of the shape
        160                 // instead of the bottom-left corner
    0   161                 var halfX = shape.Extent.X / 2.0f;
    0   162                 var halfY = shape.Extent.Y / 2.0f;
```

```
0   164                 return Matrix4.CreateTranslation(
0   165                     -halfX - camera.Offset.X,
0   166                     -halfY - camera.Offset.Y,
0   167                     0.0f) *
0   168                        Matrix4.CreateRotationZ(shape.Rotation) *
0   169                        Matrix4.CreateTranslation(shape.Position.X + halfX + camera.Offset.X,
0   170                         shape.Position.Y + halfY + camera.Offset.Y,
0   171                            0.0f);
0   172             }
    173
    174             public void Render(Shape shape, Camera camera)
0   175             {
    176
    177                 // bind this texture
0   178                 BindTexture();
    179
    180                 // render this texture
0   181                 Matrix4 modelViewMatrix = CreateMatrix(shape, camera);
0   182                 GL.MatrixMode(MatrixMode.Modelview);
0   183                 GL.LoadMatrix(ref modelViewMatrix);
    184
0   185                 GL.Translate(camera.Offset.X, camera.Offset.Y, 0);
    186                 //GL.Scale(camera.Scale, camera.Scale, 1f);
0   187                 GL.Color4(1f, 1f, 1f, 1f);
0   188                 GL.Begin(PrimitiveType.Quads);
    189
0   190                 GL.TexCoord2(0, 1); GL.Vertex2(0.0f, 0.0f);                      // Top Left
0   191                 GL.TexCoord2(0, 0); GL.Vertex2(0.0f, shape.Extent.Y);            // Bottom Left
0   192                 GL.TexCoord2(1, 0); GL.Vertex2(shape.Extent.X, shape.Extent.Y);  // Bottom Right
0   193                 GL.TexCoord2(1, 1); GL.Vertex2(shape.Extent.X, 0.0f);            // Top Right
    194
0   195                 GL.End();
    196
    197                 // unbind this texture
0   198                 UnbindTexture();
0   199             }
    200
    201             public void Render(Shape shape)
0   202             {
```

```
                // bind this texture
0               BindTexture();

                // render this texture
0               Matrix4 modelViewMatrix = CreateMatrix(shape);
0               GL.MatrixMode(MatrixMode.Modelview);
0               GL.LoadMatrix(ref modelViewMatrix);

0               GL.Color4(1f, 1f, 1f, 1f);
0               GL.Begin(PrimitiveType.Quads);

0               GL.TexCoord2(0, 1); GL.Vertex2(0.0f, 0.0f);                     // Top Left
0               GL.TexCoord2(0, 0); GL.Vertex2(0.0f, shape.Extent.Y);           // Bottom Left
0               GL.TexCoord2(1, 0); GL.Vertex2(shape.Extent.X, shape.Extent.Y); // Bottom Right
0               GL.TexCoord2(1, 1); GL.Vertex2(shape.Extent.X, 0.0f);           // Top Right

0               GL.End();

                // unbind this texture
0               UnbindTexture();
0           }
        }
    }
```

DIKUArcade.GUI.Window

Summary

Class:	DIKUArcade.GUI.Window
Assembly:	DIKUArcade
File(s):	/home/student/SU23Guest/DIKUGames/DIKUArcade/DIKUArcade/GUI/Window.cs
Covered lines:	6
Uncovered lines:	172
Coverable lines:	178
Total lines:	377
Line coverage:	3.3% (6 of 178)
Covered branches:	0
Total branches:	74
Branch coverage:	0% (0 of 74)
Covered methods:	1
Total methods:	26
Method coverage:	3.8% (1 of 26)

Metrics

Method	Branch coverage	Cyclomatic complexity	Line coverage
get_Title()	100%	1	0%
set_Title(...)	100%	1	0%
get_Resizable()	100%	1	0%
CreateOpenGLContext(	100%	1	100%
ActivateThisWindowCo	100%	1	0%
.ctor(...)	0%	10	0%
Finalize()	0%	2	0%
DefaultResizeHandler	0%	2	0%
AddDefaultResizeHand	100%	1	0%
RemoveDefaultResizeH	100%	1	0%
DefaultKeyEventHandl	0%	4	0%
AddDefaultKeyEventHa	100%	1	0%
RemoveDefaultKeyEven	100%	1	0%
SetKeyEventHandler(.	0%	2	0%
IsRunning()	100%	1	0%
CloseWindow()	100%	1	0%

DestroyWindow()	100%	1	0%
Clear()	100%	1	0%
SetClearColor(...)	0%	12	0%
SetClearColor(...)	0%	12	0%
SetClearColor(...)	0%	12	0%
SetClearColor(...)	0%	12	0%
SetClearColor(...)	100%	1	0%
SwapBuffers()	100%	1	0%
PollEvents()	100%	1	0%
SaveScreenShot()	0%	6	0%

File(s)

/home/student/SU23Guest/DIKUGames/DIKUArcade/DIKUArcade/GUI/Window.cs

#	Line	Line coverage
	1	`using System;`
	2	`using PixelFormat = System.Drawing.Imaging.PixelFormat;`
	3	`using Bitmap = System.Drawing.Bitmap;`
	4	`using RotateFlipType = System.Drawing.RotateFlipType;`
	5	`using System.Drawing.Imaging;`
	6	`using System.IO;`
	7	`using OpenTK.Windowing.Desktop;`
	8	`using OpenTK.Windowing.Common;`
	9	`using OpenTK.Graphics.OpenGL;`
	10	`using DIKUArcade.Input;`
	11	
	12	`namespace DIKUArcade.GUI {`
	13	`    /// <summary>`
	14	`    /// This class represents a graphical window in the DIKUArcade game engine.`
	15	`    /// </summary>`
	16	`    public class Window {`
	17	`        private static uint screenShotCounter;`
	18	
	19	`        /// <summary>`
	20	`        /// Every DIKUArcade.Window instance has its own private`
	21	`        /// OpenTK.GameWindow object.`
	22	`        /// </summary>`

```
        private GameWindow window;

        private bool isRunning;

        public string Title {
            get { return window.Title; }
            set { window.Title = value; }
        }

        /// <summary>
        /// Get or set if this Window instance should be resizable.
        /// </summary>
        public bool Resizable { get; set; } = true;

        /// <summary>
        /// Instance for transforming OpenTK key events to DIKUArcade-interfaced
        /// key events, based on globalization settings.
        /// </summary>
        private IKeyTransformer keyTransformer;


        #region OpenGLContext

        /// <summary>
        /// A static, private OpenTK.GameWindow instance.
        /// Only used for initializing an OpenGL context in the background.
        /// </summary>
        private static GameWindow _contextWin;

        /// <summary>
        /// Use this method to create an OpenGL context.
        /// Never use this method in your application, ONLY in unit testing.
        /// This will enable you to unit test classes which use OpenGL-dependent
        /// function calls, including ‘Text’, ‘Image’, and ‘ImageStride’ classes.
        /// </summary>
        public static void CreateOpenGLContext() {
            var settings = new GameWindowSettings();
            var nativeSettings = new NativeWindowSettings();
            Window._contextWin = new GameWindow(settings, nativeSettings);
            Window._contextWin.Context.MakeCurrent();
```

```
1    63            }
     64
     65            #endregion
     66
     67
0    68            private void ActivateThisWindowContext(string title, uint width, uint height, bool fullscreen) {
     69                // We use OpenGL 2.0 (ie. fixed-function pipeline!)
0    70                var settings = new GameWindowSettings();
0    71                settings.IsMultiThreaded = false;
0    72                var nativeSettings = new NativeWindowSettings();
0    73                nativeSettings.Profile = ContextProfile.Any;
0    74                nativeSettings.WindowState = WindowState.Normal;
0    75                nativeSettings.API = ContextAPI.OpenGL;
0    76                nativeSettings.APIVersion = new Version(2, 0);
0    77                nativeSettings.IsFullscreen = fullscreen;
     78
0    79                window = new GameWindow(settings, nativeSettings) {
0    80                    Title = title,
0    81                    Size = new OpenTK.Mathematics.Vector2i((int)width, (int)height)
0    82                };
     83
0    84                GL.ClearDepth(1);
0    85                GL.ClearColor(0.0f, 0.0f, 0.0f, 1.0f);
     86
0    87                isRunning = true;
0    88                window.Context.MakeCurrent();
0    89                window.IsVisible = true;
     90
0    91                GL.Viewport(0, 0, window.Size.X, window.Size.Y);
0    92                GL.MatrixMode(MatrixMode.Projection);
0    93                GL.LoadIdentity();
0    94                GL.Ortho(0.0,1.0,0.0,1.0, 0.0, 4.0);
0    95            }
     96
     97
0    98            public Window(WindowArgs windowArgs) {
     99                // keyboard layout
0   100                switch(windowArgs.KeyboardLayout) {
    101                    case KeyboardLayout.Danish:
0   102                        keyTransformer = new Input.Languages.DanishKeyTransformer();
```

```
0   103                     break;
    104                 default:
0   105                     throw new ArgumentException("Window(): only Danish keyboard layout is currently supported!");
    106             }
    107
    108             // window dimensions
0   109             uint width = windowArgs.Width;
0   110             uint height = windowArgs.Height;
0   111             switch (windowArgs.AspectRatio) {
    112                 case WindowAspectRatio.Aspect_Custom:
0   113                     break;
    114                 case WindowAspectRatio.Aspect_1X1:
0   115                     width = height;
0   116                     break;
    117                 case WindowAspectRatio.Aspect_3X2:
0   118                     width = (height * 3) / 2;
0   119                     break;
    120                 case WindowAspectRatio.Aspect_4X3:
0   121                     width = (height * 4) / 3;
0   122                     break;
    123                 case WindowAspectRatio.Aspect_16X9:
0   124                     width = (height * 16) / 9;
0   125                     break;
    126                 default:
0   127                     throw new ArgumentException("Window(): invalid aspect ratio!");
    128             }
    129
    130             // create and bind OpenGL context
0   131             ActivateThisWindowContext(windowArgs.Title, width, height, windowArgs.FullScreen);
    132
    133             // setup event handlers
0   134             if (windowArgs.Resizable) {
0   135                 AddDefaultResizeHandler();
0   136             }
0   137             AddDefaultKeyEventHandler();
0   138         }
    139
0   140          Window() {
0   141             if (window != null) this.DestroyWindow();
0   142         }
```

```
143
144
145             #region WINDOW_RESIZE
146
147             private void DefaultResizeHandler(ResizeEventArgs args) {
148                 if (!Resizable) {
149                     return;
150                 }
151
152                 // GL.Viewport(0, 0, window.Size.X, window.Size.Y);
153                 GL.Viewport(0, 0, args.Size.X, args.Size.Y); // TODO: Is that right?
154                 GL.MatrixMode(MatrixMode.Projection);
155                 GL.LoadIdentity();
156                 GL.Ortho(0.0, 1.0, 0.0, 1.0, 0.0, 4.0);
157             }
158
159             private void AddDefaultResizeHandler() {
160                 window.Resize += DefaultResizeHandler;
161             }
162
163             private void RemoveDefaultResizeHandler() {
164                 window.Resize -= DefaultResizeHandler;
165             }
166
167             #endregion WINDOW_RESIZE
168
169             #region KEY_EVENT_HANDLERS
170
171
172
173             private void DefaultKeyEventHandler(KeyboardKeyEventArgs args) {
174                 switch(args.Key) {
175                     case OpenTK.Windowing.GraphicsLibraryFramework.Keys.Escape:
176                         CloseWindow();
177                         break;
178                     case OpenTK.Windowing.GraphicsLibraryFramework.Keys.F12:
179                         SaveScreenShot();
180                         break;
181                     default:
182                         break;
```

```
        }
    }

    private void AddDefaultKeyEventHandler() {
        window.KeyDown += DefaultKeyEventHandler;
    }

    private void RemoveDefaultKeyEventHandler() {
        window.KeyDown -= DefaultKeyEventHandler;
    }

    /// <summary>
    /// Attach the specified keyHandler method argument to this window object.
    /// All key inputs will thereafter be directed to this keyHandler.
    /// </summary>
    public void SetKeyEventHandler(Action<KeyboardAction,KeyboardKey> keyHandler) {
        RemoveDefaultKeyEventHandler();
        window.KeyDown += args => {
            if (!args.IsRepeat) keyHandler(KeyboardAction.KeyPress, keyTransformer.TransformKey(args.Key));
        };
        window.KeyUp += args => {
            keyHandler(KeyboardAction.KeyRelease, keyTransformer.TransformKey(args.Key));
        };
    }

    #endregion KEY_EVENT_HANDLERS

    /// <summary>
    /// Check if the Window is still running.
    /// </summary>
    public bool IsRunning() {
        return isRunning;
    }

    /// <summary>
    /// Sets the window running variable to false such that calls to
    /// `IsRunning()` afterwards will return false. This will allow one
    /// to exit the game loop.
    /// </summary>
    public void CloseWindow() {
```

```
0   223                 isRunning = false;
0   224             }
    225
    226             /// <summary>
    227             /// Close the underlying OpenTK window object.
    228             /// Do not call this method outside the engine.
    229             /// </summary>
0   230             internal void DestroyWindow() {
0   231                 window.Close();
0   232                 window.Dispose();
0   233                 window = null;
0   234             }
    235
    236             /// <summary>
    237             /// Clear the Window with a uniform background color.
    238             /// </summary>
0   239             public void Clear() {
0   240                 GL.Clear(ClearBufferMask.ColorBufferBit | ClearBufferMask.DepthBufferBit);
0   241             }
    242
    243             #region SET_CLEAR_COLOR
    244
    245             /// <summary>
    246             /// Set color to be used as clear color when using the Window.Clear() method.
    247             /// </summary>
    248             /// <param name="vec">Vec3F containing the RGB color values.</param>
    249             /// <exception cref="ArgumentOutOfRangeException">Normalized color values must be
    250             /// between 0 and 1.</exception>
0   251             public void SetClearColor(Math.Vec3F vec) {
0   252                 if (vec.X < 0.0f || vec.X > 1.0f ||
0   253                     vec.Y < 0.0f || vec.Y > 1.0f ||
0   254                     vec.Z < 0.0f || vec.Z > 1.0f) {
0   255                     throw new ArgumentOutOfRangeException(
0   256                         $"Normalized RGB Color values must be between 0 and 1: {vec}");
    257                 }
0   258                 GL.ClearColor(vec.X, vec.Y, vec.Z, 1.0f);
0   259             }
    260
    261             /// <summary>
    262             /// Set color to be used as clear color when using the Window.Clear() method.
```

```
        /// </summary>
        /// <param name="vec">Vec3I containing the RGB color values.</param>
        /// <exception cref="ArgumentOutOfRangeException">Color values must be between 0 and 255.</exception>
0       public void SetClearColor(Math.Vec3I vec) {
0           if (vec.X < 0 || vec.X > 255 ||
0               vec.Y < 0 || vec.Y > 255 ||
0               vec.Z < 0 || vec.Z > 255) {
0               throw new ArgumentOutOfRangeException(
0                   $"RGB Color values must be between 0 and 255: {vec}");
            }
0           GL.ClearColor(vec.X / 255.0f, vec.Y / 255.0f, vec.Z / 255.0f, 1.0f);
0       }

        /// <summary>
        /// Set color to be used as clear color when using the Window.Clear() method.
        /// </summary>
        /// <param name="r">red channel value</param>
        /// <param name="g">green channel value</param>
        /// <param name="b">blue channel value</param>
        /// <exception cref="ArgumentOutOfRangeException">Normalized color values must be
        /// between 0 and 1.</exception>
0       public void SetClearColor(float r, float g, float b) {
0           if (r < 0.0f || r > 1.0f ||
0               g < 0.0f || g > 1.0f ||
0               b < 0.0f || b > 1.0f) {
0               throw new ArgumentOutOfRangeException(
0                   $"Normalized RGB Color values must be between 0 and 1: ({r},{g},{b})");
            }
0           GL.ClearColor(r, g, b, 1.0f);
0       }

        /// <summary>
        /// Set color to be used as clear color when using the Window.Clear() method.
        /// </summary>
        /// <param name="r">red channel value</param>
        /// <param name="g">green channel value</param>
        /// <param name="b">blue channel value</param>
        /// <exception cref="ArgumentOutOfRangeException">Color values must be between 0 and 255.</exception>
0       public void SetClearColor(int r, int g, int b) {
0           if (r < 0 || r > 255 ||
```

```
0   303                 g < 0 || g > 255 ||
0   304                 b < 0 || b > 255) {
0   305                 throw new ArgumentOutOfRangeException(
0   306                     $"RGB Color values must be between 0 and 255: ({r},{g},{b})");
    307             }
0   308             GL.ClearColor(r / 255.0f, g / 255.0f, b / 255.0f, 1.0f);
0   309         }
    310
    311         /// <summary>
    312         /// Set color to be used as clear color when using the Window.Clear() method.
    313         /// </summary>
    314         /// <param name="color">System.Drawing.Color object containing color channel values.</param>
0   315         public void SetClearColor(System.Drawing.Color color) {
0   316             SetClearColor(new Math.Vec3I(color.R, color.G, color.B));
0   317         }
    318
    319         #endregion
    320
    321         /// <summary>
    322         /// Swap double buffers for the Window.
    323         /// </summary>
0   324         public void SwapBuffers() {
0   325             window.SwapBuffers();
0   326         }
    327
    328         /// <summary>
    329         /// Check for incoming keyboard or mouse events.
    330         /// </summary>
0   331         public void PollEvents() {
0   332             window.ProcessEvents();
0   333         }
    334
    335         /// <summary>
    336         /// Save a screenshot of the Window's current state.
    337         /// </summary>
    338         /// <exception cref="GraphicsContextMissingException"></exception>
0   339         public void SaveScreenShot() {
0   340             if (window.Context == null) {
0   341                 throw new ArgumentNullException("GraphicsContextMissingException");
    342             }
```

```
0   344                 var bmp = new Bitmap(window.ClientSize.X, window.ClientSize.Y, PixelFormat.Format24bppRgb);
0   345                 var data = bmp.LockBits(new System.Drawing.Rectangle(0, 0, window.ClientSize.X, window.ClientSize.Y),
0   346                                         ImageLockMode.WriteOnly,
0   347                                         PixelFormat.Format24bppRgb);
0   348                 GL.ReadPixels(0, 0, window.ClientSize.X, window.ClientSize.Y,
0   349                     OpenTK.Graphics.OpenGL.PixelFormat.Bgr,
0   350                     PixelType.UnsignedByte, data.Scan0);
0   351                 bmp.UnlockBits(data);
    352
0   353                 bmp.RotateFlip(RotateFlipType.RotateNoneFlipY);
    354
    355                 // save screenshot, not in bin/Debug (et sim.), but in a logical place
0   356                 var dir = new DirectoryInfo(Path.GetDirectoryName(
0   357                     System.Reflection.Assembly.GetExecutingAssembly().Location));
    358
0   359                 while (dir.Name != "bin") {
0   360                     dir = dir.Parent;
0   361                 }
0   362                 dir = dir.Parent;
    363
    364                 // build the save path
0   365                 var saveName = $"screenShot_{Window.screenShotCounter++}.bmp";
0   366                 var folder = Path.Combine(dir.ToString(), "screenShots");
0   367                 var path = Path.Combine(folder, saveName);
    368
0   369                 if (!Directory.Exists(folder)) {
0   370                     Directory.CreateDirectory(folder);
0   371                 }
    372
0   373                 bmp.Save(path);
0   374                 Console.WriteLine($"Screenshot saved as: {path}");
0   375             }
    376         }
    377     }
```

DIKUArcade.GUI.WindowArgs

Summary

Class:	DIKUArcade.GUI.WindowArgs
Assembly:	DIKUArcade
File(s):	/home/student/SU23Guest/DIKUGames/DIKUArcade/DIKUArcade/GUI/WindowArgs.cs
Covered lines:	0
Uncovered lines:	7
Coverable lines:	7
Total lines:	27
Line coverage:	0% (0 of 7)
Covered branches:	0
Total branches:	0
Covered methods:	0
Total methods:	7
Method coverage:	0% (0 of 7)

Metrics

Method	Branch coverage	Cyclomatic complexity	Line coverage
get_KeyboardLayout()	100%	1	0%
get_Title()	100%	1	0%
get_Width()	100%	1	0%
get_Height()	100%	1	0%
get_AspectRatio()	100%	1	0%
get_FullScreen()	100%	1	0%
get_Resizable()	100%	1	0%

File(s)

/home/student/SU23Guest/DIKUGames/DIKUArcade/DIKUArcade/GUI/WindowArgs.cs

#	Line	Line coverage
	1	`using DIKUArcade.Input;`
	2	
	3	`namespace DIKUArcade.GUI {`
	4	`    /// <summary>`

```
        /// Arguments for constructing a DIKUArcade.Window object.
        /// Use this class to set fundamental properties of the window.
        /// </summary>
        public class WindowArgs {
            /* Globalisation settings */
0           public KeyboardLayout KeyboardLayout { get; set; } = KeyboardLayout.Danish;

            /* Basic window properties */
0           public string Title { get; set; } = "DIKUArcade";
0           public uint Width { get; set; } = 500U;
0           public uint Height { get; set; } = 500U;

            /// <summary>
            /// Specify window aspect ratio. If this value is something else than `WindowAspectRatio.Aspect_Custom`,
            /// then the width of the window will be calculated automatically based on the height.
            /// </summary>
0           public WindowAspectRatio AspectRatio { get; set; } = WindowAspectRatio.Aspect_Custom;

            /* Graphical properties */
0           public bool FullScreen { get; set; } = false;
0           public bool Resizable { get; set; } = true;
        }
    }
```

DIKUArcade.Input.Languages.DanishKeyTransformer

Summary

Class:	DIKUArcade.Input.Languages.DanishKeyTransformer
Assembly:	DIKUArcade
File(s):	3Guest/DIKUGames/DIKUArcade/DIKUArcade/Input/Languages/DanishKeyTransformer.cs
Covered lines:	0
Uncovered lines:	125
Coverable lines:	125
Total lines:	155
Line coverage:	0% (0 of 125)
Covered branches:	0
Total branches:	194
Branch coverage:	0% (0 of 194)
Covered methods:	0
Total methods:	1
Method coverage:	0% (0 of 1)

Metrics

Method	Branch coverage	Cyclomatic complexity	Line coverage
TransformKey(...)	0%	194	0%

File(s)

3Guest/DIKUGames/DIKUArcade/DIKUArcade/Input/Languages/DanishKeyTransformer.cs

#	Line	Line coverage
	1	`using OpenTK.Windowing.GraphicsLibraryFramework;`
	2	`using DIKUArcade.Input;`
	3	`using System;`
	4	
	5	`namespace DIKUArcade.Input.Languages`
	6	`{`
	7	`    /// <summary>`
	8	`    /// Represents the Danish keyboard layout.`
	9	`    /// </summary>`

```
        public class DanishKeyTransformer : IKeyTransformer
        {
            public KeyboardKey TransformKey(Keys key)
            {
                switch (key)
                {
                    case Keys.Unknown: return KeyboardKey.Unknown;
                    case Keys.Space: return KeyboardKey.Space;
                    case Keys.Apostrophe: return KeyboardKey.Danish_OE;
                    case Keys.Comma: return KeyboardKey.Comma;
                    case Keys.Minus: return KeyboardKey.Plus;
                    case Keys.Period: return KeyboardKey.Period;
                    case Keys.Slash: return KeyboardKey.Minus;

                    case Keys.D0: return KeyboardKey.Num_0;
                    case Keys.D1: return KeyboardKey.Num_1;
                    case Keys.D2: return KeyboardKey.Num_2;
                    case Keys.D3: return KeyboardKey.Num_3;
                    case Keys.D4: return KeyboardKey.Num_4;
                    case Keys.D5: return KeyboardKey.Num_5;
                    case Keys.D6: return KeyboardKey.Num_6;
                    case Keys.D7: return KeyboardKey.Num_7;
                    case Keys.D8: return KeyboardKey.Num_8;
                    case Keys.D9: return KeyboardKey.Num_9;
                    case Keys.Semicolon: return KeyboardKey.Danish_AE;
                    case Keys.Equal: return KeyboardKey.AcuteAccent;

                    case Keys.A: return KeyboardKey.A;
                    case Keys.B: return KeyboardKey.B;
                    case Keys.C: return KeyboardKey.C;
                    case Keys.D: return KeyboardKey.D;
                    case Keys.E: return KeyboardKey.E;
                    case Keys.F: return KeyboardKey.F;
                    case Keys.G: return KeyboardKey.G;
                    case Keys.H: return KeyboardKey.H;
                    case Keys.I: return KeyboardKey.I;
                    case Keys.J: return KeyboardKey.J;
                    case Keys.K: return KeyboardKey.K;
                    case Keys.L: return KeyboardKey.L;
                    case Keys.M: return KeyboardKey.M;
```

```
0    50                 case Keys.N: return KeyboardKey.N;
0    51                 case Keys.O: return KeyboardKey.O;
0    52                 case Keys.P: return KeyboardKey.P;
0    53                 case Keys.Q: return KeyboardKey.Q;
0    54                 case Keys.R: return KeyboardKey.R;
0    55                 case Keys.S: return KeyboardKey.S;
0    56                 case Keys.T: return KeyboardKey.T;
0    57                 case Keys.U: return KeyboardKey.U;
0    58                 case Keys.V: return KeyboardKey.V;
0    59                 case Keys.W: return KeyboardKey.W;
0    60                 case Keys.X: return KeyboardKey.X;
0    61                 case Keys.Y: return KeyboardKey.Y;
0    62                 case Keys.Z: return KeyboardKey.Z;
     63
0    64                 case Keys.LeftBracket: return KeyboardKey.Danish_AA;
0    65                 case Keys.Backslash: return KeyboardKey.Apostrophe;
0    66                 case Keys.RightBracket: return KeyboardKey.Diaresis;
0    67                 case Keys.GraveAccent: return KeyboardKey.FractionOneHalf;
0    68                 case Keys.Escape: return KeyboardKey.Escape;
0    69                 case Keys.Enter: return KeyboardKey.Enter;
0    70                 case Keys.Tab: return KeyboardKey.Tab;
0    71                 case Keys.Backspace: return KeyboardKey.Backspace;
0    72                 case Keys.Insert: return KeyboardKey.Insert;
0    73                 case Keys.Delete: return KeyboardKey.Delete;
     74
0    75                 case Keys.Right: return KeyboardKey.Right;
0    76                 case Keys.Left: return KeyboardKey.Left;
0    77                 case Keys.Down: return KeyboardKey.Down;
0    78                 case Keys.Up: return KeyboardKey.Up;
     79
0    80                 case Keys.PageUp: return KeyboardKey.PageUp;
0    81                 case Keys.PageDown: return KeyboardKey.PageDown;
0    82                 case Keys.Home: return KeyboardKey.Home;
0    83                 case Keys.End: return KeyboardKey.End;
     84
0    85                 case Keys.CapsLock: return KeyboardKey.CapsLock;
0    86                 case Keys.ScrollLock: return KeyboardKey.ScrollLock;
0    87                 case Keys.NumLock: return KeyboardKey.NumLock;
0    88                 case Keys.PrintScreen: return KeyboardKey.PrintScreen;
0    89                 case Keys.Pause: return KeyboardKey.Pause;
```

```
        90
0       91                  case Keys.F1: return KeyboardKey.F1;
0       92                  case Keys.F2: return KeyboardKey.F2;
0       93                  case Keys.F3: return KeyboardKey.F3;
0       94                  case Keys.F4: return KeyboardKey.F4;
0       95                  case Keys.F5: return KeyboardKey.F5;
0       96                  case Keys.F6: return KeyboardKey.F6;
0       97                  case Keys.F7: return KeyboardKey.F7;
0       98                  case Keys.F8: return KeyboardKey.F8;
0       99                  case Keys.F9: return KeyboardKey.F9;
0      100                  case Keys.F10: return KeyboardKey.F10;
0      101                  case Keys.F11: return KeyboardKey.F11;
0      102                  case Keys.F12: return KeyboardKey.F12;
0      103                  case Keys.F13: return KeyboardKey.F13;
0      104                  case Keys.F14: return KeyboardKey.F14;
0      105                  case Keys.F15: return KeyboardKey.F15;
0      106                  case Keys.F16: return KeyboardKey.F16;
0      107                  case Keys.F17: return KeyboardKey.F17;
0      108                  case Keys.F18: return KeyboardKey.F18;
0      109                  case Keys.F19: return KeyboardKey.F19;
0      110                  case Keys.F20: return KeyboardKey.F20;
0      111                  case Keys.F21: return KeyboardKey.F21;
0      112                  case Keys.F22: return KeyboardKey.F22;
0      113                  case Keys.F23: return KeyboardKey.F23;
0      114                  case Keys.F24: return KeyboardKey.F24;
0      115                  case Keys.F25: return KeyboardKey.F25;
       116
0      117                  case Keys.KeyPad0: return KeyboardKey.KeyPad0;
0      118                  case Keys.KeyPad1: return KeyboardKey.KeyPad1;
0      119                  case Keys.KeyPad2: return KeyboardKey.KeyPad2;
0      120                  case Keys.KeyPad3: return KeyboardKey.KeyPad3;
0      121                  case Keys.KeyPad4: return KeyboardKey.KeyPad4;
0      122                  case Keys.KeyPad5: return KeyboardKey.KeyPad5;
0      123                  case Keys.KeyPad6: return KeyboardKey.KeyPad6;
0      124                  case Keys.KeyPad7: return KeyboardKey.KeyPad7;
0      125                  case Keys.KeyPad8: return KeyboardKey.KeyPad8;
0      126                  case Keys.KeyPad9: return KeyboardKey.KeyPad9;
0      127                  case Keys.KeyPadDecimal: return KeyboardKey.KeyPadDecimal;
0      128                  case Keys.KeyPadDivide: return KeyboardKey.KeyPadDivide;
0      129                  case Keys.KeyPadMultiply: return KeyboardKey.KeyPadMultiply;
```

```
0   130                     case Keys.KeyPadSubtract: return KeyboardKey.KeyPadSubtract;
0   131                     case Keys.KeyPadAdd: return KeyboardKey.KeyPadAdd;
0   132                     case Keys.KeyPadEnter: return KeyboardKey.KeyPadEnter;
0   133                     case Keys.KeyPadEqual: return KeyboardKey.KeyPadEqual;
    134
0   135                     case Keys.LeftShift: return KeyboardKey.LeftShift;
0   136                     case Keys.LeftControl: return KeyboardKey.LeftControl;
0   137                     case Keys.LeftAlt: return KeyboardKey.LeftAlt;
0   138                     case Keys.LeftSuper: return KeyboardKey.LeftSuper;
0   139                     case Keys.RightShift: return KeyboardKey.RightShift;
0   140                     case Keys.RightControl: return KeyboardKey.RightControl;
0   141                     case Keys.RightAlt: return KeyboardKey.RightAlt;
0   142                     case Keys.RightSuper: return KeyboardKey.RightSuper;
0   143                     case Keys.Menu: return KeyboardKey.Menu;
    144
    145                     default:
0   146                         break;
    147                 }
    148
    149                 // special case for Danish keyboard layout, since this key is not given a
    150                 // name by OpenTK 4.5.
0   151                 if ((int)key == 161) { return KeyboardKey.LessThan; }
0   152                 else { return KeyboardKey.Unknown; }
0   153             }
    154         }
    155     }
```

DIKUArcade.Math.Vec2D

Summary

Class:	DIKUArcade.Math.Vec2D
Assembly:	DIKUArcade
File(s):	/home/student/SU23Guest/DIKUGames/DIKUArcade/DIKUArcade/Math/Vec2D.cs
Covered lines:	0
Uncovered lines:	39
Coverable lines:	39
Total lines:	61
Line coverage:	0% (0 of 39)
Covered branches:	0
Total branches:	0
Covered methods:	0
Total methods:	12
Method coverage:	0% (0 of 12)

Metrics

Method	Branch coverage	Cyclomatic complexity	Line coverage
.ctor(...)	100%	1	0%
.ctor()	100%	1	0%
op_Addition(...)	100%	1	0%
op_Subtraction(...)	100%	1	0%
op_Multiply(...)	100%	1	0%
op_Multiply(...)	100%	1	0%
op_Multiply(...)	100%	1	0%
Dot(...)	100%	1	0%
Length()	100%	1	0%
Copy()	100%	1	0%
GetHashCode()	100%	1	0%
ToString()	100%	1	0%

File(s)

/home/student/SU23Guest/DIKUGames/DIKUArcade/DIKUArcade/Math/Vec2D.cs

#	Line	Line coverage
	1	`namespace DIKUArcade.Math {`
	2	`    public class Vec2D {`
	3	`        public double X;`
	4	`        public double Y;`
	5	
0	6	`        public Vec2D(double x, double y) {`
0	7	`            X = x;`
0	8	`            Y = y;`
0	9	`        }`
	10	
0	11	`        public Vec2D() : this(0.0f, 0.0f) { }`
	12	
0	13	`        public static Vec2D operator +(Vec2D v1, Vec2D v2) {`
0	14	`            return new Vec2D(v1.X + v2.X, v1.Y + v2.Y);`
0	15	`        }`
	16	
0	17	`        public static Vec2D operator -(Vec2D v1, Vec2D v2) {`
0	18	`            return new Vec2D(v1.X - v2.X, v1.Y - v2.Y);`
0	19	`        }`
	20	
	21	`        // pairwise multiplication`
0	22	`        public static Vec2D operator *(Vec2D v1, Vec2D v2) {`
0	23	`            return new Vec2D(v1.X * v2.X, v1.Y * v2.Y);`
0	24	`        }`
	25	
0	26	`        public static Vec2D operator *(Vec2D v, double s) {`
0	27	`            return new Vec2D(v.X * s, v.Y * s);`
0	28	`        }`
	29	
0	30	`        public static Vec2D operator *(double s, Vec2D v) {`
0	31	`            return new Vec2D(v.X * s, v.Y * s);`
0	32	`        }`
	33	
0	34	`        public static double Dot(Vec2D v1, Vec2D v2) {`
0	35	`            return v1.X * v2.X + v1.Y * v2.Y;`
0	36	`        }`
	37	
0	38	`        public double Length() {`
0	39	`            return System.Math.Sqrt(X * X + Y * Y);`

```
0    40            }
     41
0    42            public Vec2D Copy() {
0    43                return new Vec2D(X, Y);
0    44            }
     45
0    46            public override int GetHashCode() {
     47                // Source: http://stackoverflow.com/a/263416/5801152
     48                unchecked // Overflow is fine, just wrap
0    49                {
0    50                    var hash = 17;
0    51                    hash = hash * 23 + X.GetHashCode();
0    52                    hash = hash * 23 + Y.GetHashCode();
0    53                    return hash;
     54                }
0    55            }
     56
0    57            public override string ToString() {
0    58                return $"Vec2D({X},{Y})";
0    59            }
     60        }
     61    }
```

DIKUArcade.Math.Vec2F

Summary

Class:	DIKUArcade.Math.Vec2F
Assembly:	DIKUArcade
File(s):	/home/student/SU23Guest/DIKUGames/DIKUArcade/DIKUArcade/Math/Vec2F.cs
Covered lines:	8
Uncovered lines:	37
Coverable lines:	45
Total lines:	70
Line coverage:	17.7% (8 of 45)
Covered branches:	0
Total branches:	0
Covered methods:	3
Total methods:	14
Method coverage:	21.4% (3 of 14)

Metrics

Method	Branch coverage	Cyclomatic complexity	Line coverage
.ctor(...)	100%	1	100%
.ctor()	100%	1	100%
op_Addition(...)	100%	1	100%
op_Subtraction(...)	100%	1	0%
op_Multiply(...)	100%	1	0%
op_Multiply(...)	100%	1	0%
op_Multiply(...)	100%	1	0%
op_Division(...)	100%	1	0%
Dot(...)	100%	1	0%
Length()	100%	1	0%
Normalize(...)	100%	1	0%
Copy()	100%	1	0%
GetHashCode()	100%	1	0%
ToString()	100%	1	0%

File(s)

/home/student/SU23Guest/DIKUGames/DIKUArcade/DIKUArcade/Math/Vec2F.cs

#	Line	Line coverage
	1	`namespace DIKUArcade.Math {`
	2	`    public class Vec2F {`
	3	`        public float X;`
	4	`        public float Y;`
	5	
96594	6	`        public Vec2F(float x, float y) {`
48297	7	`            X = x;`
48297	8	`            Y = y;`
48297	9	`        }`
	10	
5109	11	`        public Vec2F() : this(0.0f, 0.0f) { }`
	12	
122	13	`        public static Vec2F operator +(Vec2F v1, Vec2F v2) {`
122	14	`            return new Vec2F(v1.X + v2.X, v1.Y + v2.Y);`
122	15	`        }`
	16	
0	17	`        public static Vec2F operator -(Vec2F v1, Vec2F v2) {`
0	18	`            return new Vec2F(v1.X - v2.X, v1.Y - v2.Y);`
0	19	`        }`
	20	
	21	`        // pairwise multiplication`
0	22	`        public static Vec2F operator *(Vec2F v1, Vec2F v2) {`
0	23	`            return new Vec2F(v1.X * v2.X, v1.Y * v2.Y);`
0	24	`        }`
	25	
0	26	`        public static Vec2F operator *(Vec2F v, float s) {`
0	27	`            return new Vec2F(v.X * s, v.Y * s);`
0	28	`        }`
	29	
0	30	`        public static Vec2F operator *(float s, Vec2F v) {`
0	31	`            return new Vec2F(v.X * s, v.Y * s);`
0	32	`        }`
	33	
0	34	`        public static Vec2F operator /(Vec2F v, float s) {`
0	35	`            return new Vec2F(v.X / s, v.Y / s);`

```
0    36            }
     37
0    38            public static float Dot(Vec2F v1, Vec2F v2) {
0    39                return v1.X * v2.X + v1.Y * v2.Y;
0    40            }
     41
0    42            public double Length() {
0    43                return System.Math.Sqrt(X * X + Y * Y);
0    44            }
     45
0    46            public static Vec2F Normalize(Vec2F v) {
0    47                return v.Copy() * (1.0f / (float)v.Length());
0    48            }
     49
0    50            public Vec2F Copy() {
0    51                return new Vec2F(X, Y);
0    52            }
     53
     54
0    55            public override int GetHashCode() {
     56                // Source: http://stackoverflow.com/a/263416/5801152
     57                unchecked // Overflow is fine, just wrap
0    58                {
0    59                    var hash = 17;
0    60                    hash = hash * 23 + X.GetHashCode();
0    61                    hash = hash * 23 + Y.GetHashCode();
0    62                    return hash;
     63                }
0    64            }
     65
0    66            public override string ToString() {
0    67                return $"Vec2F({X},{Y})";
0    68            }
     69        }
     70    }
```

DIKUArcade.Math.Vec2I

Summary

Class:	DIKUArcade.Math.Vec2I
Assembly:	DIKUArcade
File(s):	/home/student/SU23Guest/DIKUGames/DIKUArcade/DIKUArcade/Math/Vec2I.cs
Covered lines:	0
Uncovered lines:	36
Coverable lines:	36
Total lines:	57
Line coverage:	0% (0 of 36)
Covered branches:	0
Total branches:	0
Covered methods:	0
Total methods:	11
Method coverage:	0% (0 of 11)

Metrics

Method	Branch coverage	Cyclomatic complexity	Line coverage
.ctor(...)	100%	1	0%
.ctor()	100%	1	0%
op_Addition(...)	100%	1	0%
op_Subtraction(...)	100%	1	0%
op_Multiply(...)	100%	1	0%
op_Multiply(...)	100%	1	0%
op_Multiply(...)	100%	1	0%
Dot(...)	100%	1	0%
Length()	100%	1	0%
Copy()	100%	1	0%
GetHashCode()	100%	1	0%

File(s)

/home/student/SU23Guest/DIKUGames/DIKUArcade/DIKUArcade/Math/Vec2I.cs

#	Line	Line coverage

```
namespace DIKUArcade.Math {
    public class Vec2I {
        public int X;
        public int Y;

        public Vec2I(int x, int y) {
            X = x;
            Y = y;
        }

        public Vec2I() : this(0, 0) { }

        public static Vec2I operator +(Vec2I v1, Vec2I v2) {
            return new Vec2I(v1.X + v2.X, v1.Y + v2.Y);
        }

        public static Vec2I operator -(Vec2I v1, Vec2I v2) {
            return new Vec2I(v1.X - v2.X, v1.Y - v2.Y);
        }

        // pairwise multiplication
        public static Vec2I operator *(Vec2I v1, Vec2I v2) {
            return new Vec2I(v1.X * v2.X, v1.Y * v2.Y);
        }

        public static Vec2I operator *(Vec2I v, int s) {
            return new Vec2I(v.X * s, v.Y * s);
        }

        public static Vec2I operator *(int s, Vec2I v) {
            return new Vec2I(v.X * s, v.Y * s);
        }

        public static int Dot(Vec2I v1, Vec2I v2) {
            return v1.X * v2.X + v1.Y * v2.Y;
        }

        public double Length() {
            return System.Math.Sqrt(X * X + Y * Y);
        }
```

```
0    public Vec2I Copy() {
0        return new Vec2I(X, Y);
0    }

0    public override int GetHashCode() {
         // Source: http://stackoverflow.com/a/263416/5801152
         unchecked // Overflow is fine, just wrap
0        {
0            var hash = 17;
0            hash = hash * 23 + X.GetHashCode();
0            hash = hash * 23 + Y.GetHashCode();
0            return hash;
         }
0    }
     }
}
```

DIKUArcade.Math.Vec3D

Summary

Class:	DIKUArcade.Math.Vec3D
Assembly:	DIKUArcade
File(s):	/home/student/SU23Guest/DIKUGames/DIKUArcade/DIKUArcade/Math/Vec3D.cs
Covered lines:	0
Uncovered lines:	38
Coverable lines:	38
Total lines:	60
Line coverage:	0% (0 of 38)
Covered branches:	0
Total branches:	0
Covered methods:	0
Total methods:	11
Method coverage:	0% (0 of 11)

Metrics

Method	Branch coverage	Cyclomatic complexity	Line coverage
.ctor(...)	100%	1	0%
.ctor()	100%	1	0%
op_Addition(...)	100%	1	0%
op_Subtraction(...)	100%	1	0%
op_Multiply(...)	100%	1	0%
op_Multiply(...)	100%	1	0%
op_Multiply(...)	100%	1	0%
Dot(...)	100%	1	0%
Length()	100%	1	0%
Copy()	100%	1	0%
GetHashCode()	100%	1	0%

File(s)

/home/student/SU23Guest/DIKUGames/DIKUArcade/DIKUArcade/Math/Vec3D.cs

#	Line	Line coverage

```
namespace DIKUArcade.Math {
    public class Vec3D {
        public double X;
        public double Y;
        public double Z;

        public Vec3D(double x, double y, double z) {
            X = x;
            Y = y;
            Z = z;
        }

        public Vec3D() : this(0.0f, 0.0f, 0.0f) { }

        public static Vec3D operator +(Vec3D v1, Vec3D v2) {
            return new Vec3D(v1.X + v2.X, v1.Y + v2.Y, v1.Z + v2.Z);
        }

        public static Vec3D operator -(Vec3D v1, Vec3D v2) {
            return new Vec3D(v1.X - v2.X, v1.Y - v2.Y, v1.Z - v2.Z);
        }

        // pairwise multiplication
        public static Vec3D operator *(Vec3D v1, Vec3D v2) {
            return new Vec3D(v1.X * v2.X, v1.Y * v2.Y, v1.Z * v2.Z);
        }

        public static Vec3D operator *(Vec3D v, double s) {
            return new Vec3D(v.X * s, v.Y * s, v.Z * s);
        }

        public static Vec3D operator *(double s, Vec3D v) {
            return new Vec3D(v.X * s, v.Y * s, v.Z * s);
        }

        public static double Dot(Vec3D v1, Vec3D v2) {
            return v1.X * v2.X + v1.Y * v2.Y + v1.Z * v2.Z;
        }

        public double Length() {
```

```
0    41                 return System.Math.Sqrt(X * X + Y * Y + Z * Z);
0    42             }
     43
0    44             public Vec3D Copy() {
0    45                 return new Vec3D(X, Y, Z);
0    46             }
     47
0    48             public override int GetHashCode() {
     49                 // Source: http://stackoverflow.com/a/263416/5801152
     50                 unchecked // Overflow is fine, just wrap
0    51                 {
0    52                     var hash = 17;
0    53                     hash = hash * 23 + X.GetHashCode();
0    54                     hash = hash * 23 + Y.GetHashCode();
0    55                     hash = hash * 23 + Z.GetHashCode();
0    56                     return hash;
     57                 }
0    58             }
     59         }
     60     }
```

DIKUArcade.Math.Vec3F

Summary

Class:	DIKUArcade.Math.Vec3F
Assembly:	DIKUArcade
File(s):	/home/student/SU23Guest/DIKUGames/DIKUArcade/DIKUArcade/Math/Vec3F.cs
Covered lines:	0
Uncovered lines:	38
Coverable lines:	38
Total lines:	60
Line coverage:	0% (0 of 38)
Covered branches:	0
Total branches:	0
Covered methods:	0
Total methods:	11
Method coverage:	0% (0 of 11)

Metrics

Method	Branch coverage	Cyclomatic complexity	Line coverage
.ctor(...)	100%	1	0%
.ctor()	100%	1	0%
op_Addition(...)	100%	1	0%
op_Subtraction(...)	100%	1	0%
op_Multiply(...)	100%	1	0%
op_Multiply(...)	100%	1	0%
op_Multiply(...)	100%	1	0%
Dot(...)	100%	1	0%
Length()	100%	1	0%
Copy()	100%	1	0%
GetHashCode()	100%	1	0%

File(s)

/home/student/SU23Guest/DIKUGames/DIKUArcade/DIKUArcade/Math/Vec3F.cs

#	Line	Line coverage

```
namespace DIKUArcade.Math {
    public class Vec3F {
        public float X;
        public float Y;
        public float Z;

        public Vec3F(float x, float y, float z) {
            X = x;
            Y = y;
            Z = z;
        }

        public Vec3F() : this(0.0f, 0.0f, 0.0f) { }

        public static Vec3F operator +(Vec3F v1, Vec3F v2) {
            return new Vec3F(v1.X + v2.X, v1.Y + v2.Y, v1.Z + v2.Z);
        }

        public static Vec3F operator -(Vec3F v1, Vec3F v2) {
            return new Vec3F(v1.X - v2.X, v1.Y - v2.Y, v1.Z - v2.Z);
        }

        // pairwise multiplication
        public static Vec3F operator *(Vec3F v1, Vec3F v2) {
            return new Vec3F(v1.X * v2.X, v1.Y * v2.Y, v1.Z * v2.Z);
        }

        public static Vec3F operator *(Vec3F v, float s) {
            return new Vec3F(v.X * s, v.Y * s, v.Z * s);
        }

        public static Vec3F operator *(float s, Vec3F v) {
            return new Vec3F(v.X * s, v.Y * s, v.Z * s);
        }

        public static float Dot(Vec3F v1, Vec3F v2) {
            return v1.X * v2.X + v1.Y * v2.Y + v1.Z * v2.Z;
        }

        public double Length() {
```

```
0    41                 return System.Math.Sqrt(X * X + Y * Y + Z * Z);
0    42             }
     43
0    44             public Vec3F Copy() {
0    45                 return new Vec3F(X, Y, Z);
0    46             }
     47
0    48             public override int GetHashCode() {
     49                 // Source: http://stackoverflow.com/a/263416/5801152
     50                 unchecked // Overflow is fine, just wrap
0    51                 {
0    52                     var hash = 17;
0    53                     hash = hash * 23 + X.GetHashCode();
0    54                     hash = hash * 23 + Y.GetHashCode();
0    55                     hash = hash * 23 + Z.GetHashCode();
0    56                     return hash;
     57                 }
0    58             }
     59         }
     60     }
```

DIKUArcade.Math.Vec3I

Summary

Class:	DIKUArcade.Math.Vec3I
Assembly:	DIKUArcade
File(s):	/home/student/SU23Guest/DIKUGames/DIKUArcade/DIKUArcade/Math/Vec3I.cs
Covered lines:	5
Uncovered lines:	33
Coverable lines:	38
Total lines:	60
Line coverage:	13.1% (5 of 38)
Covered branches:	0
Total branches:	0
Covered methods:	1
Total methods:	11
Method coverage:	9% (1 of 11)

Metrics

Method	Branch coverage	Cyclomatic complexity	Line coverage
.ctor(...)	100%	1	100%
.ctor()	100%	1	0%
op_Addition(...)	100%	1	0%
op_Subtraction(...)	100%	1	0%
op_Multiply(...)	100%	1	0%
op_Multiply(...)	100%	1	0%
op_Multiply(...)	100%	1	0%
Dot(...)	100%	1	0%
Length()	100%	1	0%
Copy()	100%	1	0%
GetHashCode()	100%	1	0%

File(s)

/home/student/SU23Guest/DIKUGames/DIKUArcade/DIKUArcade/Math/Vec3I.cs

#	Line	Line coverage

```
        1  namespace DIKUArcade.Math {
        2      public class Vec3I {
        3          public int X;
        4          public int Y;
        5          public int Z;
        6
     1786  7          public Vec3I(int x, int y, int z) {
      893  8              X = x;
      893  9              Y = y;
      893 10              Z = z;
      893 11          }
       12
        0 13          public Vec3I() : this(0, 0, 0) { }
       14
        0 15          public static Vec3I operator +(Vec3I v1, Vec3I v2) {
        0 16              return new Vec3I(v1.X + v2.X, v1.Y + v2.Y, v1.Z + v2.Z);
        0 17          }
       18
        0 19          public static Vec3I operator -(Vec3I v1, Vec3I v2) {
        0 20              return new Vec3I(v1.X - v2.X, v1.Y - v2.Y, v1.Z - v2.Z);
        0 21          }
       22
       23          // pairwise multiplication
        0 24          public static Vec3I operator *(Vec3I v1, Vec3I v2) {
        0 25              return new Vec3I(v1.X * v2.X, v1.Y * v2.Y, v1.Z * v2.Z);
        0 26          }
       27
        0 28          public static Vec3I operator *(Vec3I v, int s) {
        0 29              return new Vec3I(v.X * s, v.Y * s, v.Z * s);
        0 30          }
       31
        0 32          public static Vec3I operator *(int s, Vec3I v) {
        0 33              return new Vec3I(v.X * s, v.Y * s, v.Z * s);
        0 34          }
       35
        0 36          public static int Dot(Vec3I v1, Vec3I v2) {
        0 37              return v1.X * v2.X + v1.Y * v2.Y + v1.Z * v2.Z;
        0 38          }
       39
        0 40          public double Length() {
```

```
0    41                 return System.Math.Sqrt(X * X + Y * Y + Z * Z);
0    42             }
     43
0    44             public Vec3I Copy() {
0    45                 return new Vec3I(X, Y, Z);
0    46             }
     47
0    48             public override int GetHashCode() {
     49                 // Source: http://stackoverflow.com/a/263416/5801152
     50                 unchecked // Overflow is fine, just wrap
0    51                 {
0    52                     var hash = 17;
0    53                     hash = hash * 23 + X.GetHashCode();
0    54                     hash = hash * 23 + Y.GetHashCode();
0    55                     hash = hash * 23 + Z.GetHashCode();
0    56                     return hash;
     57                 }
0    58             }
     59         }
     60     }
```

DIKUArcade.Math.Vec4D

Summary

Class:	DIKUArcade.Math.Vec4D
Assembly:	DIKUArcade
File(s):	/home/student/SU23Guest/DIKUGames/DIKUArcade/DIKUArcade/Math/Vec4D.cs
Covered lines:	0
Uncovered lines:	40
Coverable lines:	40
Total lines:	63
Line coverage:	0% (0 of 40)
Covered branches:	0
Total branches:	0
Covered methods:	0
Total methods:	11
Method coverage:	0% (0 of 11)

Metrics

Method	Branch coverage	Cyclomatic complexity	Line coverage
.ctor(...)	100%	1	0%
.ctor()	100%	1	0%
op_Addition(...)	100%	1	0%
op_Subtraction(...)	100%	1	0%
op_Multiply(...)	100%	1	0%
op_Multiply(...)	100%	1	0%
op_Multiply(...)	100%	1	0%
Dot(...)	100%	1	0%
Length()	100%	1	0%
Copy()	100%	1	0%
GetHashCode()	100%	1	0%

File(s)

/home/student/SU23Guest/DIKUGames/DIKUArcade/DIKUArcade/Math/Vec4D.cs

#	Line	Line coverage

```
namespace DIKUArcade.Math {
    public class Vec4D {
        public double W;
        public double X;
        public double Y;
        public double Z;

        public Vec4D(double x, double y, double z, double w) {
            X = x;
            Y = y;
            Z = z;
            W = w;
        }

        public Vec4D() : this(0.0f, 0.0f, 0.0f, 0.0f) { }

        public static Vec4D operator +(Vec4D v1, Vec4D v2) {
            return new Vec4D(v1.X + v2.X, v1.Y + v2.Y, v1.Z + v2.Z, v1.W + v2.W);
        }

        public static Vec4D operator -(Vec4D v1, Vec4D v2) {
            return new Vec4D(v1.X - v2.X, v1.Y - v2.Y, v1.Z - v2.Z, v1.W - v2.W);
        }

        // pairwise multiplication
        public static Vec4D operator *(Vec4D v1, Vec4D v2) {
            return new Vec4D(v1.X * v2.X, v1.Y * v2.Y, v1.Z * v2.Z, v1.W * v2.W);
        }

        public static Vec4D operator *(Vec4D v, double s) {
            return new Vec4D(v.X * s, v.Y * s, v.Z * s, v.W * s);
        }

        public static Vec4D operator *(double s, Vec4D v) {
            return new Vec4D(v.X * s, v.Y * s, v.Z * s, v.W * s);
        }

        public static double Dot(Vec4D v1, Vec4D v2) {
            return v1.X * v2.X + v1.Y * v2.Y + v1.Z * v2.Z + v1.W * v2.W;
        }
```

```
        41
0       42            public double Length() {
0       43                return System.Math.Sqrt(X * X + Y * Y + Z * Z + W * W);
0       44            }
        45
0       46            public Vec4D Copy() {
0       47                return new Vec4D(X, Y, Z, W);
0       48            }
        49
0       50            public override int GetHashCode() {
        51                // Source: http://stackoverflow.com/a/263416/5801152
        52                unchecked // Overflow is fine, just wrap
0       53                {
0       54                    var hash = 17;
0       55                    hash = hash * 23 + X.GetHashCode();
0       56                    hash = hash * 23 + Y.GetHashCode();
0       57                    hash = hash * 23 + Z.GetHashCode();
0       58                    hash = hash * 23 + W.GetHashCode();
0       59                    return hash;
        60                }
0       61            }
        62        }
        63    }
```

DIKUArcade.Math.Vec4F

Summary

Class:	DIKUArcade.Math.Vec4F
Assembly:	DIKUArcade
File(s):	/home/student/SU23Guest/DIKUGames/DIKUArcade/DIKUArcade/Math/Vec4F.cs
Covered lines:	0
Uncovered lines:	40
Coverable lines:	40
Total lines:	63
Line coverage:	0% (0 of 40)
Covered branches:	0
Total branches:	0
Covered methods:	0
Total methods:	11
Method coverage:	0% (0 of 11)

Metrics

Method	Branch coverage	Cyclomatic complexity	Line coverage
.ctor(...)	100%	1	0%
.ctor()	100%	1	0%
op_Addition(...)	100%	1	0%
op_Subtraction(...)	100%	1	0%
op_Multiply(...)	100%	1	0%
op_Multiply(...)	100%	1	0%
op_Multiply(...)	100%	1	0%
Dot(...)	100%	1	0%
Length()	100%	1	0%
Copy()	100%	1	0%
GetHashCode()	100%	1	0%

File(s)

/home/student/SU23Guest/DIKUGames/DIKUArcade/DIKUArcade/Math/Vec4F.cs

#	Line	Line coverage

```
namespace DIKUArcade.Math {
    public class Vec4F {
        public float W;
        public float X;
        public float Y;
        public float Z;

        public Vec4F(float x, float y, float z, float w) {
            X = x;
            Y = y;
            Z = z;
            W = w;
        }

        public Vec4F() : this(0.0f, 0.0f, 0.0f, 0.0f) { }

        public static Vec4F operator +(Vec4F v1, Vec4F v2) {
            return new Vec4F(v1.X + v2.X, v1.Y + v2.Y, v1.Z + v2.Z, v1.W + v2.W);
        }

        public static Vec4F operator -(Vec4F v1, Vec4F v2) {
            return new Vec4F(v1.X - v2.X, v1.Y - v2.Y, v1.Z - v2.Z, v1.W - v2.W);
        }

        // pairwise multiplication
        public static Vec4F operator *(Vec4F v1, Vec4F v2) {
            return new Vec4F(v1.X * v2.X, v1.Y * v2.Y, v1.Z * v2.Z, v1.W * v2.W);
        }

        public static Vec4F operator *(Vec4F v, float s) {
            return new Vec4F(v.X * s, v.Y * s, v.Z * s, v.W * s);
        }

        public static Vec4F operator *(float s, Vec4F v) {
            return new Vec4F(v.X * s, v.Y * s, v.Z * s, v.W * s);
        }

        public static float Dot(Vec4F v1, Vec4F v2) {
            return v1.X * v2.X + v1.Y * v2.Y + v1.Z * v2.Z + v1.W * v2.W;
        }
```

```
        41
0       42            public double Length() {
0       43                return System.Math.Sqrt(X * X + Y * Y + Z * Z + W * W);
0       44            }
        45
0       46            public Vec4F Copy() {
0       47                return new Vec4F(X, Y, Z, W);
0       48            }
        49
0       50            public override int GetHashCode() {
        51                // Source: http://stackoverflow.com/a/263416/5801152
        52                unchecked // Overflow is fine, just wrap
0       53                {
0       54                    var hash = 17;
0       55                    hash = hash * 23 + X.GetHashCode();
0       56                    hash = hash * 23 + Y.GetHashCode();
0       57                    hash = hash * 23 + Z.GetHashCode();
0       58                    hash = hash * 23 + W.GetHashCode();
0       59                    return hash;
        60                }
0       61            }
        62        }
        63    }
```

DIKUArcade.Math.Vec4I

Summary

Class:	DIKUArcade.Math.Vec4I
Assembly:	DIKUArcade
File(s):	/home/student/SU23Guest/DIKUGames/DIKUArcade/DIKUArcade/Math/Vec4I.cs
Covered lines:	0
Uncovered lines:	40
Coverable lines:	40
Total lines:	63
Line coverage:	0% (0 of 40)
Covered branches:	0
Total branches:	0
Covered methods:	0
Total methods:	11
Method coverage:	0% (0 of 11)

Metrics

Method	Branch coverage	Cyclomatic complexity	Line coverage
.ctor(...)	100%	1	0%
.ctor()	100%	1	0%
op_Addition(...)	100%	1	0%
op_Subtraction(...)	100%	1	0%
op_Multiply(...)	100%	1	0%
op_Multiply(...)	100%	1	0%
op_Multiply(...)	100%	1	0%
Dot(...)	100%	1	0%
Length()	100%	1	0%
Copy()	100%	1	0%
GetHashCode()	100%	1	0%

File(s)

/home/student/SU23Guest/DIKUGames/DIKUArcade/DIKUArcade/Math/Vec4I.cs

#	Line	Line coverage

```
namespace DIKUArcade.Math {
    public class Vec4I {
        public int W;
        public int X;
        public int Y;
        public int Z;

        public Vec4I(int x, int y, int z, int w) {
            X = x;
            Y = y;
            Z = z;
            W = w;
        }

        public Vec4I() : this(0, 0, 0, 0) { }

        public static Vec4I operator +(Vec4I v1, Vec4I v2) {
            return new Vec4I(v1.X + v2.X, v1.Y + v2.Y, v1.Z + v2.Z, v1.W + v2.W);
        }

        public static Vec4I operator -(Vec4I v1, Vec4I v2) {
            return new Vec4I(v1.X - v2.X, v1.Y - v2.Y, v1.Z - v2.Z, v1.W - v2.W);
        }

        // pairwise multiplication
        public static Vec4I operator *(Vec4I v1, Vec4I v2) {
            return new Vec4I(v1.X * v2.X, v1.Y * v2.Y, v1.Z * v2.Z, v1.W * v2.W);
        }

        public static Vec4I operator *(Vec4I v, int s) {
            return new Vec4I(v.X * s, v.Y * s, v.Z * s, v.W * s);
        }

        public static Vec4I operator *(int s, Vec4I v) {
            return new Vec4I(v.X * s, v.Y * s, v.Z * s, v.W * s);
        }

        public static int Dot(Vec4I v1, Vec4I v2) {
            return v1.X * v2.X + v1.Y * v2.Y + v1.Z * v2.Z + v1.W * v2.W;
        }
```

```
        41
0       42            public double Length() {
0       43                return System.Math.Sqrt(X * X + Y * Y + Z * Z + W * W);
0       44            }
        45
0       46            public Vec4I Copy() {
0       47                return new Vec4I(X, Y, Z, W);
0       48            }
        49
0       50            public override int GetHashCode() {
        51                // Source: http://stackoverflow.com/a/263416/5801152
        52                unchecked // Overflow is fine, just wrap
0       53                {
0       54                    var hash = 17;
0       55                    hash = hash * 23 + X.GetHashCode();
0       56                    hash = hash * 23 + Y.GetHashCode();
0       57                    hash = hash * 23 + Z.GetHashCode();
0       58                    hash = hash * 23 + W.GetHashCode();
0       59                    return hash;
        60                }
0       61            }
        62        }
        63    }
```

DIKUArcade.Physics.CollisionData

Summary

Class:	DIKUArcade.Physics.CollisionData
Assembly:	DIKUArcade
File(s):	ome/student/SU23Guest/DIKUGames/DIKUArcade/DIKUArcade/Physics/CollisionData.cs
Covered lines:	3
Uncovered lines:	0
Coverable lines:	3
Total lines:	23
Line coverage:	100% (3 of 3)
Covered branches:	0
Total branches:	0
Covered methods:	3
Total methods:	3
Method coverage:	100% (3 of 3)

Metrics

Method	Branch coverage	Cyclomatic complexity	Line coverage
get_Collision()	100%	1	100%
get_DirectionFactor(	100%	1	100%
get_CollisionDir()	100%	1	100%

File(s)

ome/student/SU23Guest/DIKUGames/DIKUArcade/DIKUArcade/Physics/CollisionData.cs

```
    #  Line   Line coverage
       1      using DIKUArcade.Math;
       2
       3      namespace DIKUArcade.Physics {
       4          public class CollisionData {
       5              /// <summary>
       6              /// Indicating whether or not a collision has occured.
       7              /// </summary>
 1902  8              public bool Collision { get; set; }
```

```
            /// <summary>
            /// This factor should be multiplied onto the actor shape's
            /// direction vector to get the closest position to the
            /// incident object.
            /// </summary>
960         public Vec2F DirectionFactor { get; set; }

            /// <summary>
            /// The surface normal of the incident object, indicating
            /// from which direction a collision has occured.
            /// </summary>
1040        public CollisionDirection CollisionDir { get; set; } // might sometimes be useful!
        }
    }
```

DIKUArcade.Physics.CollisionDetection

Summary

Class:	DIKUArcade.Physics.CollisionDetection
Assembly:	DIKUArcade
File(s):	tudent/SU23Guest/DIKUGames/DIKUArcade/DIKUArcade/Physics/CollisionDetection.cs
Covered lines:	84
Uncovered lines:	27
Coverable lines:	111
Total lines:	184
Line coverage:	75.6% (84 of 111)
Covered branches:	33
Total branches:	48
Branch coverage:	68.7% (33 of 48)
Covered methods:	1
Total methods:	2
Method coverage:	50% (1 of 2)

Metrics

Method	Branch coverage	Cyclomatic complexity	Line coverage
Aabb_C(...)	100%	1	0%
Aabb(...)	68.75%	48	77.06%

File(s)

tudent/SU23Guest/DIKUGames/DIKUArcade/DIKUArcade/Physics/CollisionDetection.cs

#	Line	Line coverage
	1	`using System;`
	2	`using DIKUArcade.Entities;`
	3	`using DIKUArcade.Math;`
	4	
	5	`namespace DIKUArcade.Physics {`
	6	
	7	`    // https://learnopengl.com/In-Practice/2D-Game/Collisions/Collision-detection`
	8	`    // Potentially allow for AABB with circles.`

```
        // Or use SAT.
        public class CollisionDetection {
0           public static CollisionData Aabb_C(DynamicShape actor, Shape shape) {
0               throw new NotImplementedException("CollisionDetection.Aabb_C is not finished!");

                /*
                var data = new CollisionData {
                    Collision = false,
                    DirectionFactor = new Vec2F(1.0f, 1.0f),
                    CollisionDir = CollisionDirection.CollisionDirUnchecked
                };

                var circRadius = shape.Extent.Y/2;
                var circCenter = new Vec2F(shape.Position.X + shape.Extent.X/2, shape.Position.Y + shape.Extent.Y/2);

                var staLowerLeft = new Vec2F(shape.Position.X, shape.Position.Y);
                var staUpperRight = new Vec2F(shape.Position.X + shape.Extent.X,
                    shape.Position.Y + shape.Extent.Y);
                var staCenter = new Vec2F(shape.Position.X + shape.Extent.X/2, shape.Position.Y + shape.Extent.Y/2);

                var D = circCenter - staCenter;

                // Clamp D to width/2 height/2 and add it to staCenter

                return data;
                */
            }

942         public static CollisionData Aabb(DynamicShape actor, Shape shape) {
942             var data = new CollisionData {
942                 Collision = false,
942                 DirectionFactor = new Vec2F(1.0f, 1.0f),
942                 CollisionDir = CollisionDirection.CollisionDirUnchecked
942             };

942             var dynLowerLeft = new Vec2F(actor.Position.X, actor.Position.Y);
942             var dynUpperRight = new Vec2F(actor.Position.X + actor.Extent.X,
942                 actor.Position.Y + actor.Extent.Y);

942             var staLowerLeft = new Vec2F(shape.Position.X, shape.Position.Y);
```

```
942  49                 var staUpperRight = new Vec2F(shape.Position.X + shape.Extent.X,
942  50                     shape.Position.Y + shape.Extent.Y);
     51
     52                 // inactive movement in both x- and y-direction
942  53                 if(System.Math.Abs(actor.Direction.X) < 1e-6f && System.Math.Abs(actor.Direction.Y) < 1e-6f) {
  0  54                     return data;
     55                 }
     56
     57                 // inactive movement in x-direction
942  58                 else if(System.Math.Abs(actor.Direction.X) < 1e-6f)
 92  59                 {
     60                     float entryDistanceY, exitDistanceY;
 92  61                     if(actor.Direction.Y < 0.0f)
 91  62                     {
 91  63                         entryDistanceY = staUpperRight.Y - dynLowerLeft.Y;
 91  64                         exitDistanceY = staLowerLeft.Y - dynUpperRight.Y;
 91  65                         data.CollisionDir = CollisionDirection.CollisionDirDown;
 91  66                     }
     67                     else
  1  68                     {
  1  69                         entryDistanceY = staLowerLeft.Y - dynUpperRight.Y;
  1  70                         exitDistanceY = staUpperRight.Y - dynLowerLeft.Y;
  1  71                         data.CollisionDir = CollisionDirection.CollisionDirUp;
  1  72                     }
     73
 92  74                     var entryTimeY = entryDistanceY / actor.Direction.Y;
 92  75                     var exitTimeY = exitDistanceY / actor.Direction.Y;
     76
 92  77                     bool xOverlaps = staUpperRight.X > dynLowerLeft.X && staLowerLeft.X < dynUpperRight.X;
     78
 92  79                     if(entryTimeY < exitTimeY && entryTimeY >= 0.0f && entryTimeY < 1.0f && xOverlaps)
 15  80                     {
 15  81                         data.DirectionFactor.Y = entryTimeY;
 15  82                         data.Collision = true;
 15  83                         return data;
     84                     }
     85                     else
 77  86                     {
 77  87                         return data;
     88                     }
```

```
                }
                // inactive movement in y-direction
                else if(System.Math.Abs(actor.Direction.Y) < 1e-6f)
                {
                    float entryDistanceX, exitDistanceX;
                    if(actor.Direction.X < 0.0f)
                    {
                        entryDistanceX = staUpperRight.X - dynLowerLeft.X;
                        exitDistanceX = staLowerLeft.X - dynUpperRight.X;
                        data.CollisionDir = CollisionDirection.CollisionDirLeft;
                    }
                    else
                    {
                        entryDistanceX = staLowerLeft.X - dynUpperRight.X;
                        exitDistanceX = staUpperRight.X - dynLowerLeft.X;
                        data.CollisionDir = CollisionDirection.CollisionDirRight;
                    }

                    float entryTimeX = entryDistanceX / actor.Direction.X;
                    float exitTimeX = exitDistanceX / actor.Direction.X;

                    bool yOverlaps = staUpperRight.Y > dynLowerLeft.Y && staLowerLeft.Y < dynUpperRight.Y;

                    if(entryTimeX < exitTimeX && entryTimeX >= 0.0f && entryTimeX < 1.0f && yOverlaps)
                    {
                        data.DirectionFactor.X = entryTimeX;
                        data.Collision = true;
                    }
                    return data;
                }
                // active movement in both x- and y-direction
                else
                {
                    var entryDistance = new Vec2F();
                    var exitDistance = new Vec2F();

                    if(actor.Direction.X < 0.0f)
                    {
                        entryDistance.X = staUpperRight.X - dynLowerLeft.X;
                        exitDistance.X = staLowerLeft.X - dynUpperRight.X;
```

```
138  129                    }
     130                    else
712  131                    {
712  132                        entryDistance.X = staLowerLeft.X - dynUpperRight.X;
712  133                        exitDistance.X = staUpperRight.X - dynLowerLeft.X;
712  134                    }
850  135                    if(actor.Direction.Y < 0.0f)
157  136                    {
157  137                        entryDistance.Y = staUpperRight.Y - dynLowerLeft.Y;
157  138                        exitDistance.Y = staLowerLeft.Y - dynUpperRight.Y;
157  139                    }
     140                    else
693  141                    {
693  142                        entryDistance.Y = staLowerLeft.Y - dynUpperRight.Y;
693  143                        exitDistance.Y = staUpperRight.Y - dynLowerLeft.Y;
693  144                    }
     145
850  146                    var entryTime = new Vec2F(entryDistance.X / actor.Direction.X, entryDistance.Y / actor.Direction.Y);
850  147                    var exitTime = new Vec2F(exitDistance.X / actor.Direction.X, exitDistance.Y / actor.Direction.Y);
     148
850  149                    float entryTimeMax = System.Math.Max(entryTime.X, entryTime.Y);
850  150                    float exitTimeMin = System.Math.Min(exitTime.X, exitTime.Y);
     151
850  152                    if(entryTimeMax < exitTimeMin && (entryTime.X >= 0.0f || entryTime.Y >= 0.0f) &&
850  153                       entryTime.X < 1.0f && entryTime.Y < 1.0f)
  3  154                    {
  3  155                        if (entryTime.X > entryTime.Y)
  2  156                        {
  2  157                            data.DirectionFactor.X = entryTimeMax;
  4  158                            if (actor.Direction.X < 0.0f) {
  2  159                                data.CollisionDir = CollisionDirection.CollisionDirRight;
  2  160                            } else {
  0  161                                data.CollisionDir = CollisionDirection.CollisionDirLeft;
  0  162                            }
  2  163                        }
     164                        else
  1  165                        {
  1  166                            data.DirectionFactor.Y = entryTimeMax;
  1  167                            if (actor.Direction.Y < 0.0f) {
  0  168                                data.CollisionDir = CollisionDirection.CollisionDirUp;
```

```
1    169                            } else {
1    170                                data.CollisionDir = CollisionDirection.CollisionDirDown;
1    171                            }
1    172                        }
     173
3    174                        data.Collision = true;
3    175                        return data;
     176                    }
     177                    else
847  178                    {
847  179                        return data;
     180                    }
     181                }
942  182            }
     183        }
     184    }
```

DIKUArcade.Timers.GameTimer

Summary

Class:	DIKUArcade.Timers.GameTimer
Assembly:	DIKUArcade
File(s):	/home/student/SU23Guest/DIKUGames/DIKUArcade/DIKUArcade/Timers/GameTimer.cs
Covered lines:	0
Uncovered lines:	57
Coverable lines:	57
Total lines:	100
Line coverage:	0% (0 of 57)
Covered branches:	0
Total branches:	12
Branch coverage:	0% (0 of 12)
Covered methods:	0
Total methods:	8
Method coverage:	0% (0 of 8)

Metrics

Method	Branch coverage	Cyclomatic complexity	Line coverage
get_CapturedUpdates(	100%	1	0%
get_CapturedFrames()	100%	1	0%
.ctor()	100%	1	0%
.ctor(...)	0%	4	0%
MeasureTime()	100%	1	0%
ShouldUpdate()	0%	2	0%
ShouldRender()	0%	4	0%
ShouldReset()	0%	2	0%

File(s)

/home/student/SU23Guest/DIKUGames/DIKUArcade/DIKUArcade/Timers/GameTimer.cs

#	Line	Line coverage
	1	`using System;`
	2	`using System.Diagnostics;`

```
  3
  4  namespace DIKUArcade.Timers {
  5      public class GameTimer {
  6          private double lastTime;
  7          private double timer;
  8          private double updateTimeLimit;
  9          private double renderTimeLimit;
 10          private double deltaUpdateTime;
 11          private double deltaRenderTime;
 12          private double nowTime;
 13
 14          /// <summary>
 15          /// Get the last observed UPS count
 16          /// </summary>
0 17          public int CapturedUpdates { get; private set; }
 18          /// <summary>
 19          /// Get the last observed FPS count
 20          /// </summary>
0 21          public int CapturedFrames { get; private set; }
 22
 23          private int updates;
 24          private int frames;
 25
 26          private int desiredMaxFPS;
 27
 28          private Stopwatch stopwatch;
 29
0 30          public GameTimer() : this(30, 30) {}
 31
0 32          public GameTimer(int ups, int fps = 0) {
0 33              if (ups < 0 || fps < 0) {
0 34                  throw new ArgumentOutOfRangeException(
0 35                      $"GameTimer must have positive count values: (ups={ups},fps={fps})");
 36              }
0 37              desiredMaxFPS = fps;
 38
0 39              stopwatch = new Stopwatch();
0 40              stopwatch.Start();
 41
0 42              updateTimeLimit = 1.0 / ups;
```

```
0    43                 renderTimeLimit = 1.0 / fps;
0    44                 lastTime = stopwatch.ElapsedMilliseconds / 1000.0; // elapsed seconds
0    45                 deltaUpdateTime = 0.0;
0    46                 deltaRenderTime = 0.0;
0    47                 nowTime = 0.0;
0    48                 timer = lastTime;
     49
0    50                 frames = 0;
0    51                 updates = 0;
0    52                 CapturedFrames = 0;
0    53                 CapturedUpdates = 0;
0    54             }
     55
0    56             public void MeasureTime() {
0    57                 nowTime = stopwatch.ElapsedMilliseconds / 1000.0;
0    58                 deltaUpdateTime += (nowTime - lastTime) / updateTimeLimit;
0    59                 deltaRenderTime += (nowTime - lastTime) / renderTimeLimit;
0    60                 lastTime = nowTime;
0    61             }
     62
0    63             public bool ShouldUpdate() {
0    64                 var ret = deltaUpdateTime >= 1.0;
0    65                 if (ret) {
0    66                     updates++;
0    67                     deltaUpdateTime--;
0    68                 }
0    69                 return ret;
0    70             }
     71
0    72             public bool ShouldRender() {
0    73                 if (desiredMaxFPS < 1) {
0    74                     return true;
     75                 }
0    76                 var ret = deltaRenderTime >= 1.0;
0    77                 if (ret) {
0    78                     frames++;
0    79                     deltaRenderTime--;
0    80                 }
0    81                 return ret;
0    82             }
```

```
0    83
     84            /// <summary>
     85            /// The timer will reset if 1 second has passed.
     86            /// This information can be used to update game logic in any way desireable.
     87            /// </summary>
0    88            public bool ShouldReset() {
0    89                var ret = (stopwatch.ElapsedMilliseconds / 1000.0) - timer > 1.0;
0    90                if (ret) {
0    91                    timer += 1.0;
0    92                    CapturedUpdates = updates;
0    93                    CapturedFrames = frames;
0    94                    updates = 0;
0    95                    frames = 0;
0    96                }
0    97                return ret;
0    98            }
     99        }
    100    }
```

DIKUArcade.Timers.StaticTimer

Summary

Class:	DIKUArcade.Timers.StaticTimer
Assembly:	DIKUArcade
File(s):	/home/student/SU23Guest/DIKUGames/DIKUArcade/DIKUArcade/Timers/StaticTimer.cs
Covered lines:	8
Uncovered lines:	21
Coverable lines:	29
Total lines:	59
Line coverage:	27.5% (8 of 29)
Covered branches:	0
Total branches:	4
Branch coverage:	0% (0 of 4)
Covered methods:	2
Total methods:	7
Method coverage:	28.5% (2 of 7)

Metrics

Method	Branch coverage	Cyclomatic complexity	Line coverage
.cctor()	100%	1	100%
GetElapsedMillisecon	100%	1	100%
GetElapsedSeconds()	100%	1	0%
GetElapsedMinutes()	100%	1	0%
RestartTimer()	100%	1	0%
PauseTimer()	0%	2	0%
ResumeTimer()	0%	2	0%

File(s)

/home/student/SU23Guest/DIKUGames/DIKUArcade/DIKUArcade/Timers/StaticTimer.cs

#	Line	Line coverage
	1	`using System.Diagnostics;`
	2	
	3	`namespace DIKUArcade.Timers {`

```
        4
        5       /// <summary>
        6       /// Static timer initialized on engine startup. Can be used for
        7       /// animations based on static, discrete time intervals.
        8       /// </summary>
        9       public class StaticTimer {
       10           private static Stopwatch timer;
       11           private static bool paused;
       12
 1     13           static StaticTimer() {
 1     14               StaticTimer.timer = new Stopwatch();
 1     15               StaticTimer.timer.Start();
 1     16               StaticTimer.paused = false;
 1     17           }
       18
       19           /// <summary>
       20           /// Get the number of elapsed milliseconds since application start
       21           /// </summary>
51     22           public static long GetElapsedMilliseconds() {
51     23               return StaticTimer.timer.ElapsedMilliseconds;
51     24           }
       25
       26           /// <summary>
       27           /// Get the number of elapsed seconds since application start
       28           /// </summary>
 0     29           public static double GetElapsedSeconds() {
 0     30               return StaticTimer.timer.ElapsedMilliseconds / 1000.0;
 0     31           }
       32
       33           /// <summary>
       34           /// Get the number of elapsed minutes since application start
       35           /// </summary>
       36           /// <returns></returns>
 0     37           public static double GetElapsedMinutes() {
 0     38               return StaticTimer.timer.ElapsedMilliseconds / 60000.0;
 0     39           }
       40
 0     41           public static void RestartTimer() {
 0     42               StaticTimer.timer.Restart();
 0     43           }
```

```
        44
0       45            public static void PauseTimer() {
0       46                if (!StaticTimer.paused) {
0       47                    StaticTimer.timer.Stop();
0       48                    StaticTimer.paused = true;
0       49                }
0       50            }
        51
0       52            public static void ResumeTimer() {
0       53                if (StaticTimer.paused) {
0       54                    StaticTimer.timer.Start();
0       55                    StaticTimer.paused = false;
0       56                }
0       57            }
        58        }
        59    }
```

DIKUArcade.Timers.TimePeriod

Summary

Class:	DIKUArcade.Timers.TimePeriod
Assembly:	DIKUArcade
File(s):	/home/student/SU23Guest/DIKUGames/DIKUArcade/DIKUArcade/Timers/TimePeriod.cs
Covered lines:	8
Uncovered lines:	8
Coverable lines:	16
Total lines:	30
Line coverage:	50% (8 of 16)
Covered branches:	1
Total branches:	6
Branch coverage:	16.6% (1 of 6)
Covered methods:	3
Total methods:	5
Method coverage:	60% (3 of 5)

Metrics

Method	Branch coverage	Cyclomatic complexity	Line coverage
.ctor(...)	100%	1	100%
NewMilliseconds(...)	0%	2	0%
NewSeconds(...)	50.0%	2	100%
NewMinutes(...)	0%	2	0%
ToMilliseconds()	100%	1	100%

File(s)

/home/student/SU23Guest/DIKUGames/DIKUArcade/DIKUArcade/Timers/TimePeriod.cs

#	Line	Line coverage
	1	`using System;`
	2	
	3	`namespace DIKUArcade.Timers {`
	4	`    public struct TimePeriod {`
	5	

```
            // A TimeSpan will internally be represented as an amount of milliseconds.
            private readonly System.Int64 value;

            private TimePeriod(System.Int64 value) {
                this.value = value;
            }

            public static TimePeriod NewMilliseconds(System.Int64 value) {
                if (value < 0) { throw new System.ArgumentOutOfRangeException("value cannot be negative."); }
                return new TimePeriod(value);
            }

            public static TimePeriod NewSeconds(System.Double value) {
                if (value < 0.0) { throw new System.ArgumentOutOfRangeException("value cannot be negative."); }
                return new TimePeriod((System.Int64)System.Math.Floor(value * 1000));
            }

            public static TimePeriod NewMinutes(System.Double value) {
                if (value < 0.0) { throw new System.ArgumentOutOfRangeException("value cannot be negative."); }
                return new TimePeriod((System.Int64)System.Math.Floor(value * 60000));
            }

            public System.Int64 ToMilliseconds() => value;
        }
    }
```

6	9
6	10
6	11
0	13
0	14
0	15
0	16
6	18
6	19
6	20
6	21
0	23
0	24
0	25
0	26
200	28

DIKUArcade.Utilities.FileIO

Summary

Class:	DIKUArcade.Utilities.FileIO
Assembly:	DIKUArcade
File(s):	/home/student/SU23Guest/DIKUGames/DIKUArcade/DIKUArcade/Utilities/FileIO.cs
Covered lines:	9
Uncovered lines:	0
Coverable lines:	9
Total lines:	23
Line coverage:	100% (9 of 9)
Covered branches:	2
Total branches:	2
Branch coverage:	100% (2 of 2)
Covered methods:	1
Total methods:	1
Method coverage:	100% (1 of 1)

Metrics

Method	Branch coverage	Cyclomatic complexity	Line coverage
GetProjectPath()	100%	2	100%

File(s)

/home/student/SU23Guest/DIKUGames/DIKUArcade/DIKUArcade/Utilities/FileIO.cs

#	Line	Line coverage
	1	`using System.IO;`
	2	
	3	`namespace DIKUArcade.Utilities {`
	4	`    public class FileIO {`
	5	
	6	`        /// <summary>`
	7	`        /// Return the platform-specific path of the current project directory`
	8	`        /// </summary>`
	9	`        /// <returns></returns>`

```
1   10          public static string GetProjectPath() {
    11              // find base path
1   12              var dir = new DirectoryInfo(Path.GetDirectoryName(
1   13                  System.Reflection.Assembly.GetExecutingAssembly().Location));
    14
5   15              while (dir.Name != "bin") {
2   16                  dir = dir.Parent;
2   17              }
1   18              dir = dir.Parent;
    19
1   20              return dir.FullName.ToString();
1   21          }
    22      }
    23  }
```

DIKUArcade.Utilities.RandomGenerator

Summary

Class:	DIKUArcade.Utilities.RandomGenerator
Assembly:	DIKUArcade
File(s):	student/SU23Guest/DIKUGames/DIKUArcade/DIKUArcade/Utilities/RandomGenerator.cs
Covered lines:	0
Uncovered lines:	6
Coverable lines:	6
Total lines:	13
Line coverage:	0% (0 of 6)
Covered branches:	0
Total branches:	2
Branch coverage:	0% (0 of 2)
Covered methods:	0
Total methods:	2
Method coverage:	0% (0 of 2)

Metrics

Method	Branch coverage	Cyclomatic complexity	Line coverage
get_Generator()	100%	1	0%
.cctor()	0%	2	0%

File(s)

student/SU23Guest/DIKUGames/DIKUArcade/DIKUArcade/Utilities/RandomGenerator.cs

#	Line	Line coverage
	1	`using System;`
	2	
	3	`namespace DIKUArcade.Utilities {`
	4	`    public class RandomGenerator {`
0	5	`        public static Random Generator { get; private set; }`
	6	
0	7	`        static RandomGenerator() {`
0	8	`            if (RandomGenerator.Generator == null) {`

```
0     9                    RandomGenerator.Generator = new Random();
0    10                }
0    11            }
     12        }
     13    }
```