



KAUNO TECHNOLOGIJOS UNIVERSITETAS

INFORMATIKOS FAKULTETAS

TAIKOMOSIOS INFORMATIKOS KATEDRA

**OBJEKTINIS PROGRAMŲ PROJEKTAVIMAS
(T120B516)**

Laboratorinis darbas Nr. 1

Atliko:

Paulius Ratkevičius IFF 8/12

Dovydas Bražas IFF-8/6

Antanas Stepanauskas IFF-8/6

Priėmė:

Andrej Ušaniov

KAUNAS, 2021

TURINYS

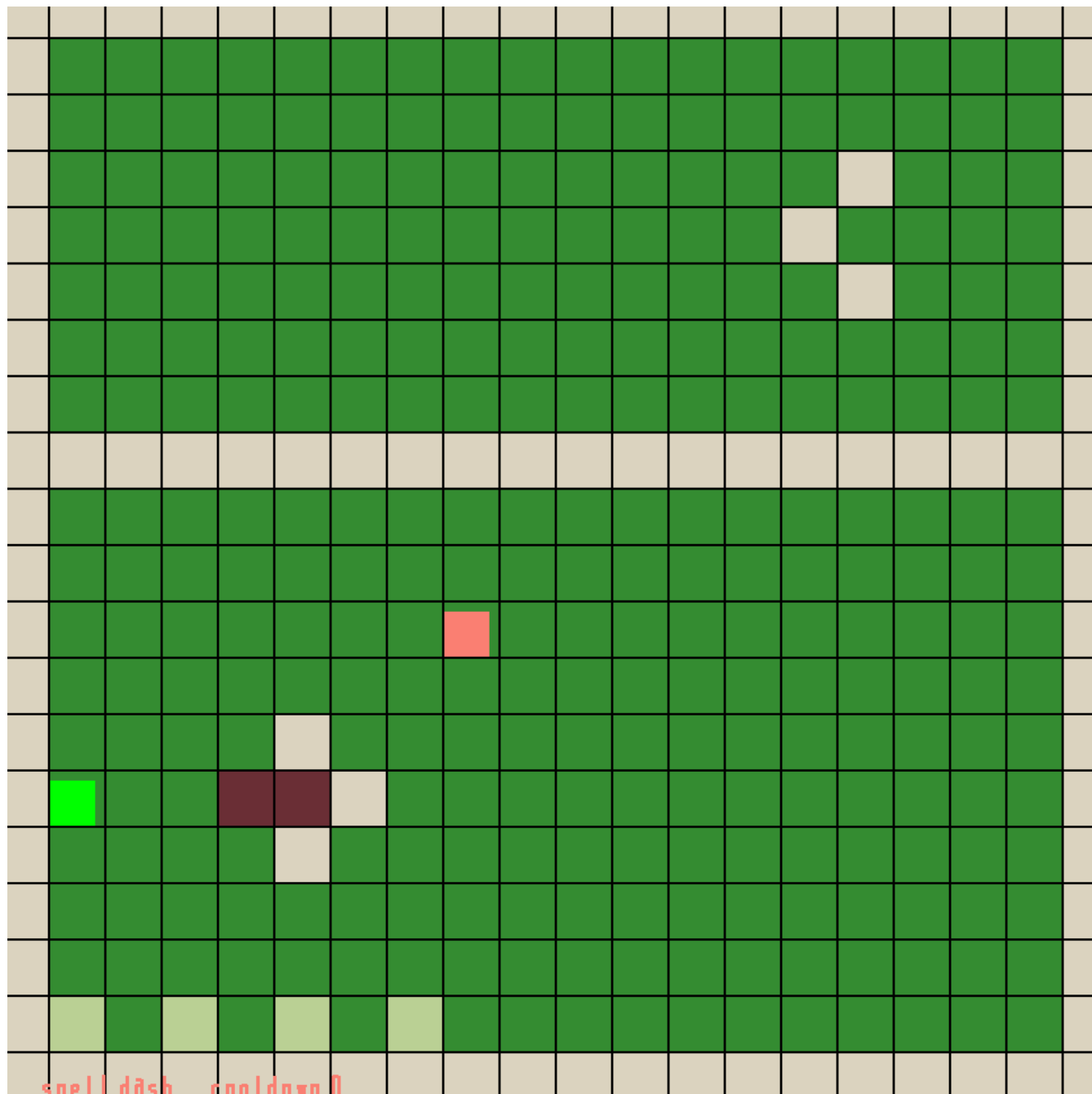
Contents

TURINYS	1
1. PROJEKTO APRAŠYMAS	4
1.1 pav. Žaidimo prototipo nuotrauka	4
2. ŽAIDIMO REIKALAVIMAI	5
2.1. ŽAIDIMO LYGIAI	5
2.2. PIRMASIS LYGIS	5
2.3. ANTRAS LYGIS.....	5
2.4. TREČIAS LYGIS	5
3. PROJEKTUI NAUDOTOS TECHNOLOGIJOS	6
4. USE CASE DIAGRAMA	6
4.2 pav. Use case diagrama	6
5. KLASIŲ DIAGRAMA	7
5.3 pav. Klasijų diagrama.....	7
5.1. ŠABLONAI	8
5.1.1. Singleton	8
5.4 pav. Singleton diagrama	8
5.1.2. Strategy	19
5.5 pav. Strategy diagrama	19
5.1.3. Decorator.....	30
5.6 pav. Decorator diagrama	30
5.1.4. Abstract Factory	41
5.7 pav. Abstract Factory diagrama.....	41
5.1.5. Observer.....	64
5.8 pav. Observer diagrama.....	64
5.1.6. Facade	79
5.10 pav. Facade diagrama	79
5.1.7. Factory	105

5.11	pav. Factory diagrama	105
5.12	pav. Builder diagrama	109
5.13	pav. Prototype diagrama	121
6.	PROJEKTO APRAŠYMAS ANTRA DALIS	122
6.1	pav. Žaidimo prototipo nuotrauka	122
7.	ŽAIDIMO REIKALAVIMAI	123
	ŽAIDIMO LYGIAI	123
	PIRMASIS LYGIS	123
	ANTRAS LYGIS	123
	TREČIAS LYGIS	123
8.	PROJEKTUI NAUDOTOS TECHNOLOGIJOS	124
9.	USE CASE DIAGRAMA	124
	4.2 pav. Use case diagrama	124
10.	KLASIŲ DIAGRAMA	125
	5.14 pav. Klasių diagrama	125
11	ŠABLONAI	126
	GameState	126
	• Žaidimo laukimas	126
	• Žaidimo vykdymas	126
	• Žaidimo užbaigimo bandymas	126
	• Žaidimo lentos pakeitimas į kitą	126
	UML diagrama:	126
	Kodas:	
	Proxy šablonas	163
	UML diagrama:	188
	Kodas:	
	Interpreter	202
	Chain of responsibility	214
	Composite	223

1. PROJEKTO APRAŠYMAS

Žaidimas, kuriame strategiškai dėlioji bombas, kuriomis gali sunaikinti kliūtis bei priešus, išvenginėji spąstų bei naudoji įgytas galias kad įgautum pranašumą prieš savo varžovą.



1.1 pav. Žaidimo prototipo nuotrauka

2. ŽAIDIMO REIKALAVIMAI

2.1. ŽAIDIMO LYGIAI

Žaidimas susidės iš trijų lygių, kurie vienas nuo kito skirsis savo žaidimo strategijomis bei sudėtingumu.

2.2. PIRMASIS LYGIS

Jame bus sukurtas pasaulis, kuris susidės iš sienų, kurių kiaurai pereiti negalima, bet galima jas susprogdinti naudojant bombas, bei žemės, per kurią žaidėjas gales laisvai vaikščioti. Žaidimo tikslas nugalėti savo priešininką, taktiškai naikinant sienas. Žaidėjai turės po 3 gyvybes, viena gyvybė yra prarandama jeigu savo arba priešininko bomba sprogs šalia.

2.3. ANTRAS LYGIS

Antrame lygyje atsiranda nauja kliūtis- tai spąstai, kurie sugeneruojami sukuriant 2 lygio pasaulį. Visi spąstai atrodo taip pat tik žaidėjai nežino, ką jis gali padaryti, tik žino tą kad visi spąstai jį užsaldys. Taip pat spąstai gali žaidėja sulėtinti, nuimti gyvybę ar jį kažkur nukelti.

2.4. TREČIAS LYGIS

Trečiame lygyje atsiranda žaidėjo įgudis. Kai 3 lygio pasaulis yra sugeneruotas kas 10 sekundžių atsiranda žaidėjo įgudis, ant kurio užlipus žaidėjas gauna viena iš 4 pagerinimų:

- Sulėtinimas priešininko. Galima naudoti kas 30 sekundžių.
- Nusikeltas į naują vietą. Galima naudoti kas 20 sekundžių.
- Sienų peršokimas. Galima naudoti kas 15 sekundžių.
- Greitas žaidėjo paslinkimas. Galima naudoti kas 15 sekundžių.

3. PROJEKTUI NAUDOTOS TECHNOLOGIJOS

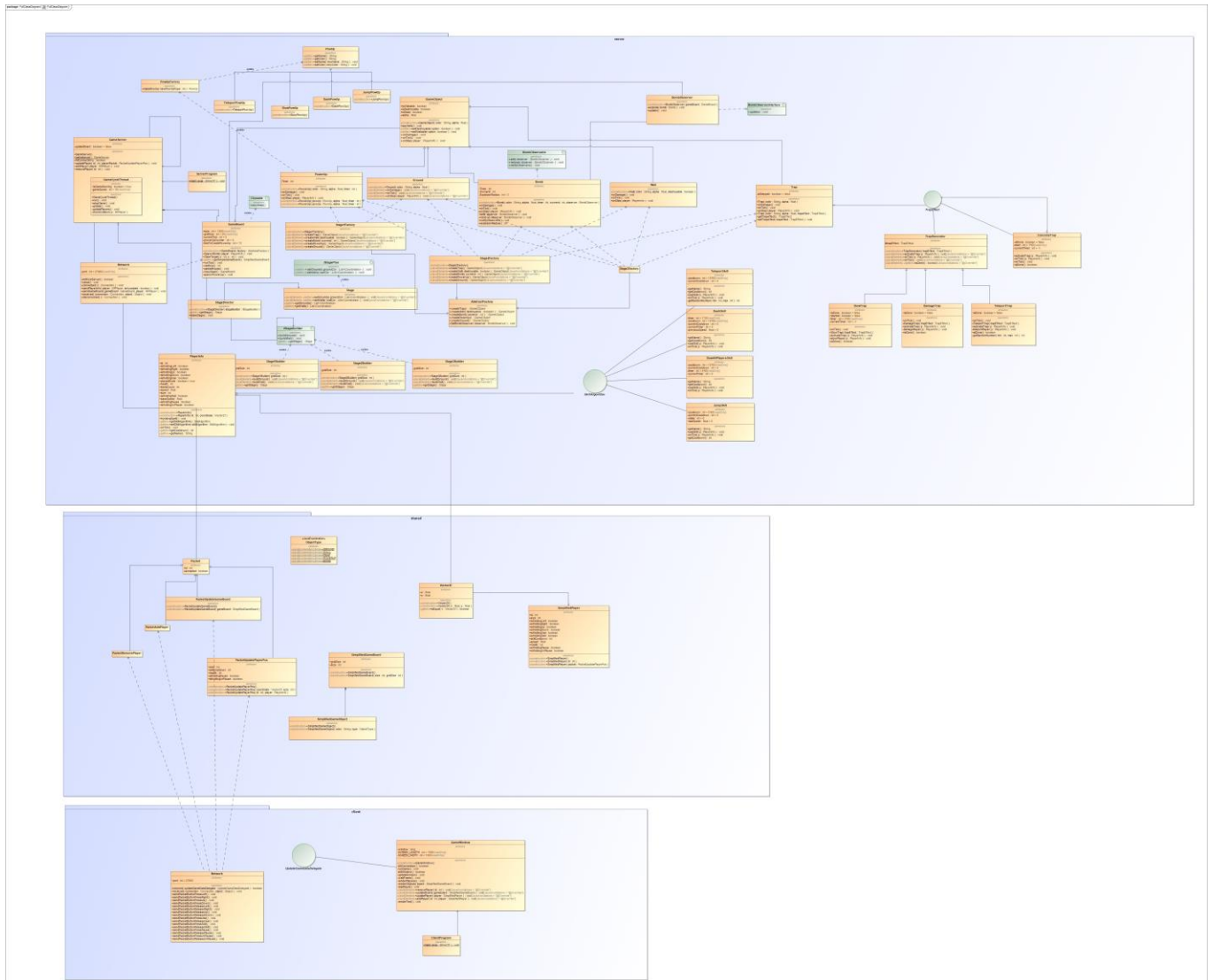
Serverio ir kliento komunikacijai naudotas Kryonet. Programos lango sukūrimui ir piešimui buvo pasitelkta lwjgl ir OpenGL 1.1 įrankiai.

4. USE CASE DIAGRAMA



4.2 pav. Use case diagrama

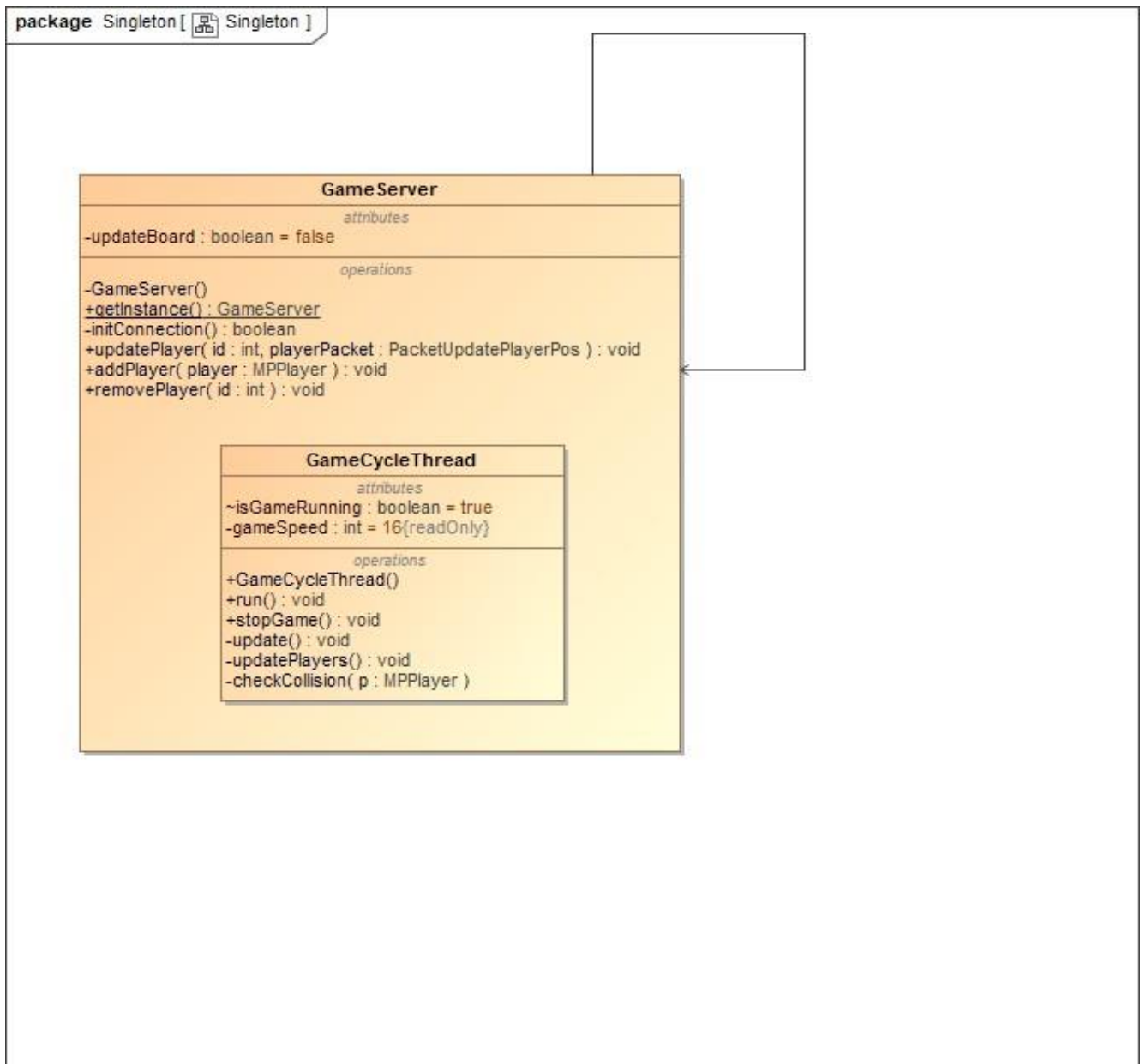
5. KLASIŲ DIAGRAMA



5.3 pav. Kласių diagrama

5.1. ŠABLONAI

5.1.1. Singleton



5.4 pav. Singleton diagrama

Šis algoritmas buvo pasirinktas dėl to, jog vienu metu gali vykti tik vienas žaidimas su bendra logika. Taip pat šią žaidimo klasę paranku pasiekti visame projekte.

```

1 package server;
2
3 import java.util.HashMap;
4 import java.util.Map;
5
6 import shared.Vector2f;
7 import shared.PacketUpdatePlayerPos;
8
9 class GameServer
10 {
11     private static GameServer gameServer = null;
12
13     protected volatile GameBoard gameBoard;
14     protected volatile Map<Integer, MPPlayer> players;
15     protected volatile Network network;
16     private GameCycleThread thread;
17     private Stage1Factory stage1factory;
18     private boolean updateBoard = false;
19
20     private GameServer()
21     {
22         //Init Connection
23         if (!initConnection())
24         {
25             System.err.println("ERROR Connecting to host");
26             return;
27         }
28
29         this.players = new HashMap<Integer, MPPlayer>();

```

```
30     this.stage1factory = new Stage1Factory();
31     this.gameBoard = new GameBoard(stage1factory);
32
33     this.thread = new GameCycleThread();
34     this.thread.start();
35 }
36
37 public static GameServer getInstance()
38 {
39     if (gameServer == null)
40     {
41         gameServer = new GameServer();
42     }
43     return gameServer;
44 }
45
46 private boolean initConnection()
47 {
48     this.network = new Network();
49
50     if (!this.network.initKryoServer())
51     {
52         return false;
53     }
54
55     return true;
56 }
57
58 public void updatePlayer(int id, PacketUpdatePlayerPos playerPacket)
```

```

59 {
60     MPPlayer player = players.get(id);
61     if (player != null)
62     {
63         player.isHoldingUp = playerPacket.isHoldingUp != null ?
playerPacket.isHoldingUp : player.isHoldingUp;
64         player.isHoldingDown = playerPacket.isHoldingDown != null ?
playerPacket.isHoldingDown : player.isHoldingDown;
65         player.isHoldingLeft = playerPacket.isHoldingLeft != null ?
playerPacket.isHoldingLeft : player.isHoldingLeft;
66         player.isHoldingRight = playerPacket.isHoldingRight != null ?
playerPacket.isHoldingRight : player.isHoldingRight;
67         player.isHoldingUse = playerPacket.isHoldingUse != null ?
playerPacket.isHoldingUse : player.isHoldingUse;
68         player.isHoldingSkill = playerPacket.isHoldingSkill != null ?
playerPacket.isHoldingSkill : player.isHoldingSkill;
69         players.put(player.id, player);
70     }
71 }
72
73 public void addPlayer(MPPlayer player)
74 {
75     this.players.put(player.c.getID(), player);
76     this.network.sendGameBoard(gameBoard, player);
77
78 }
79
80 public void removePlayer(int id)
81 {

```

```
82
83     players.remove(id);
84 }
85
86
87 private class GameCycleThread extends Thread
88 {
89     volatile boolean isGameRunning = true;
90     private final int gameSpeed = 16; //The lower the number the
    faster the game is
91
92     public GameCycleThread()
93     {
94         this.isGameRunning = true;
95     }
96
97     public void run()
98     {
99         while (this.isGameRunning)
100         {
101             try
102             {
103                 //Probably should use Timer instead
104                 Thread.sleep(gameSpeed);
105                 this.update();
106             }
107             catch (InterruptedException e)
108             {
109                 e.printStackTrace();
110             }
111         }
112     }
113 }
```

```
110         this . stopGame () ;
111     }
112 }
113 }
114
115 public void stopGame ()
116 {
117     this . isGameRunning = false ;
118
119     //network should probably be closed by the parent
120     network . close () ;
121 }
122
123 private void update ()
124 {
125     updatePlayers () ;
126     gameBoard . runTick () ;
127 }
128
129 private void updatePlayers ()
130 {
131     for (MPPlayer p : players . values ())
132     {
133         if (p . isHoldingPause )
134         {
135
136         }
137
138         if (p . isHoldingSkill)
```

```
139         {
140             p.tryUsingSpell();
141         }
142
143         p.onTick();
144
145         if (p.isHoldingUse)
146         {
147             gameBoard.SpawnBomb(p);
148         }
149
150         p.coordinate = checkCollision(p);
151
152         network.sendGameBoard(gameBoard, p);
153         network.sendPlayerInfo(p, true);
154     }
155 }
156
157 private Vector2f checkCollision(MPPlayer p)
158 {
159
160     Vector2f coordsAfterMove = new Vector2f(p.coordinate.x, p.
coordinate.y);
161
162     float padding = 0.001f;
163     float cellSize = gameBoard.cellSize();
164
165     boolean moveX = true;
166     boolean moveY = true;
```

```

167
168     if (p.isHoldingLeft)
169     {
170         coordsAfterMove.x -= p.speed;
171     }
172
173     if (p.isHoldingRight)
174     {
175
176         coordsAfterMove.x += p.speed;
177     }
178
179     boolean collidingLeft = ((int)coordsAfterMove.x / cellSize -
padding) < ((int)p.coordinate.x / cellSize);
180     boolean collidingRight = (((int)coordsAfterMove.x + p.size +
padding) / cellSize) > (((int)p.coordinate.x + p.size) / cellSize);
181     boolean isCollidingX = collidingLeft || collidingRight;
182     moveX = !(coordsAfterMove.x <= 0 || coordsAfterMove.x >=
gameBoard.size - p.size);
183     //Some smoothing when going around edges would be nice
184     if (isCollidingX && moveX)
185     {
186         int x = 0, y = 0, y1 = 0;
187         if (collidingLeft)
188         {
189             x = (int) ((coordsAfterMove.x) / cellSize);
190             y = (int) (p.coordinate.y / cellSize);
191             y1 = (int) ((p.coordinate.y + p.size) / cellSize);
192

```



```

193     }
194     if (collidingRight)
195     {
196         x = (int) ((coordsAfterMove.x + p.size) / cellSize);
197         y = (int) (p.coordinate.y / cellSize);
198         y1 = (int) ((p.coordinate.y + p.size) / cellSize);
199
200     }
201
202     if (y == y1)
203     {
204         moveX = gameBoard.objects[x][y].isWalkable ;
205         gameBoard.objects[x][y].onStep(p);
206     }
207     else
208     {
209         moveX = gameBoard.objects[x][y].isWalkable && gameBoard.
objects[x][y1].isWalkable ;
210         gameBoard.objects[x][y].onStep(p);
211         gameBoard.objects[x][y1].onStep(p);
212     }
213 }
214
215
216 if (p.isHoldingUp)
217 {
218     coordsAfterMove.y += p.speed ;
219 }
220

```

```

221     if (p.isHoldingDown)
222     {
223         coordsAfterMove.y -= p.speed;
224     }
225
226     boolean collidingUp = (((int)coordsAfterMove.y + p.size - padding
227 ) / cellSize) > (((int)p.coordinate.y + p.size) / cellSize);
228     boolean collidingDown = ((int)coordsAfterMove.y / cellSize +
229 padding) < ((int)p.coordinate.y / cellSize);
230
231     boolean isCollidingY = collidingUp || collidingDown;
232     moveY = !(coordsAfterMove.y <= 0 || coordsAfterMove.y >=
233 gameBoard.size - p.size);
234
235     //Some smoothing when going around edges would be nice
236     if (isCollidingY && moveY)
237     {
238         int x = 0, x1 = 0, y = 0;
239
240         if (collidingUp)
241         {
242             y = (int) ((coordsAfterMove.y + p.size) / cellSize);
243             x = (int) (p.coordinate.x / cellSize);
244             x1 = (int) ((p.coordinate.x + p.size) / cellSize);
245
246             }
247
248         if (collidingDown)
249         {
250             y = (int) ((coordsAfterMove.y) / cellSize);
251             x = (int) (p.coordinate.x / cellSize);

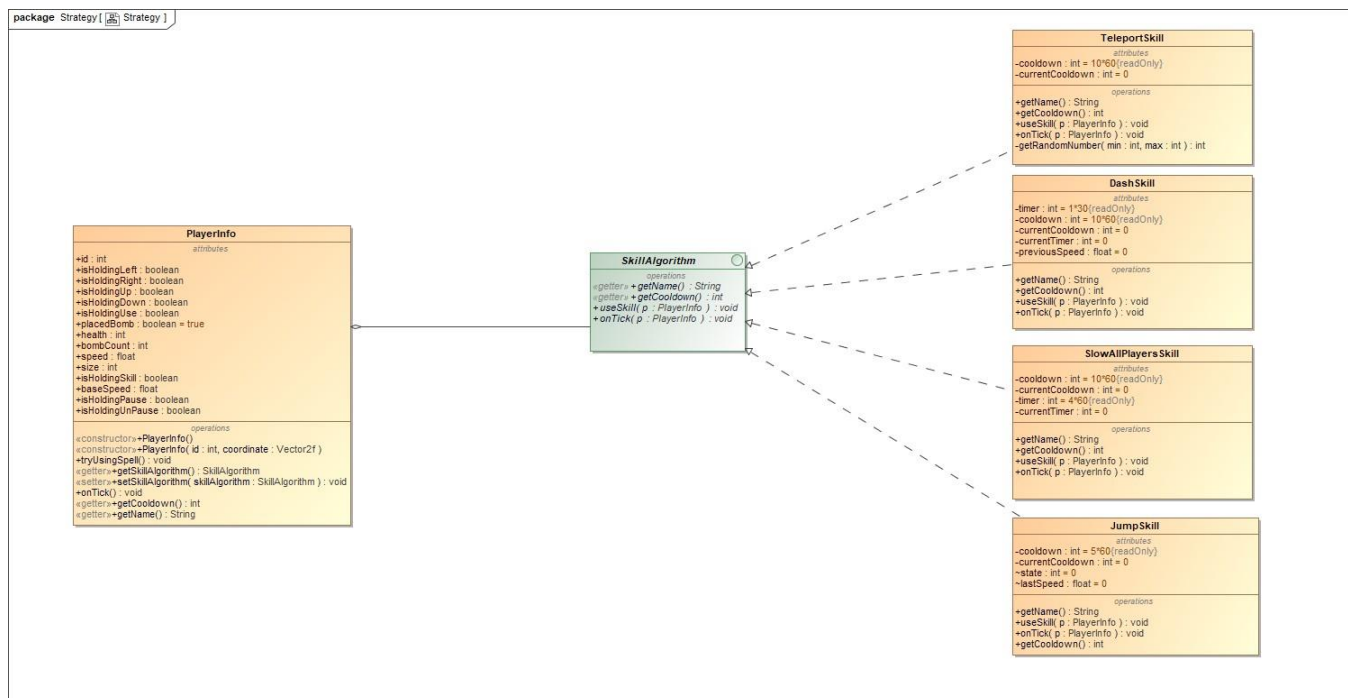
```

```

247         x1 = (int) ((p.coordinate.x + p.size) / cellSize);
248     }
249
250     if (x == x1)
251     {
252         moveY = gameBoard.objects[x][y].isWalkable;
253         gameBoard.objects[x][y].onStep(p);
254     }
255     else
256     {
257         moveY = gameBoard.objects[x][y].isWalkable && gameBoard.
objects[x1][y].isWalkable;
258         gameBoard.objects[x][y].onStep(p);
259         gameBoard.objects[x1][y].onStep(p);
260     }
261 }
262
263 coordsAfterMove.x = moveX ? coordsAfterMove.x : p.coordinate.x;
264 coordsAfterMove.y = moveY ? coordsAfterMove.y : p.coordinate.y;
265
266 return coordsAfterMove;
267
268 }
269
270 }
271
272 }

```

5.1.2. Strategy



5.5 pav. Strategy diagrama

Strategy šablono poreikis atsirado norint įterpti skirtingą funkcionalumą žaidėjui nepakeičiant kaip jis jį naudoja. Pasitelkti skirtingi algoritmai skirtingiems įgudžiams.

```

1 package server;
2
3 public interface SkillAlgorithm
4 {
5     int getCooldown();
6     void useSkill(PlayerInfo p);
7     void onTick(PlayerInfo p);
8     String getName();
9 }
  
```

```

1 package server;
2
  
```

```

3 public class JumpSkill implements SkillAlgorithm
4 {
5     private final int cooldown = 5 * 60; //5 seconds
6     private final String name = "Jump";
7     private int currentCooldown = 0;
8     int state = 0;
9     float lastSpeed = 0;
10    @Override
11    public void useSkill(PlayerInfo p)
12    {
13        if (this.currentCooldown == 0)
14        {
15            this.state = 2;
16            this.currentCooldown = this.cooldown;
17        }
18    }
19
20    @Override
21    public void onTick(PlayerInfo p)
22    {
23        if (this.currentCooldown > 0)
24        {
25            this.currentCooldown--;
26        }
27
28        if (this.state == 2)
29        {
30            GameServer.gameserver = GameServer.getInstance();
31            this.lastSpeed = p.speed;

```

```

32     p.speed = (gameserver.gameBoard.size / gameserver.gameBoard.
gridSize) * 2;
33     this.state = 1;
34 }
35 else if (this.state == 1)
36 {
37     p.speed = lastSpeed;
38     this.state = 0;
39 }
40 }
41
42 @Override
43 public String getName()
44 {
45     return this.name;
46 }
47
48 @Override
49 public int getCooldown()
50 {
51     return this.currentCooldown;
52 }
53 }

```

```

1 package server;
2
3 import java.util.ArrayList;
4 import java.util.HashMap;
5 import java.util.List;

```

```

6 import java.util.Map;
7
8 import shared.SimplifiedPlayer;
9
10 public class SlowAllPlayersSkill implements SkillAlgorithm
11 {
12     private final int cooldown = 10 * 60; //10 seconds
13     private final String name = "Slow";
14     private int currentCooldown = 0;
15     private final int timer = 4 * 60; //4 seconds
16     private int currentTimer = 0;
17     protected volatile Map<Integer, SimplifiedPlayer> simplifiedPlayers;
18     @Override
19     public void useSkill(PlayerInfo p)
20     {
21         if (this.currentCooldown == 0)
22         {
23             GameServer gameserver = GameServer.getInstance();
24             simplifiedPlayers = new HashMap<Integer, SimplifiedPlayer>();
25             for(MPPlayer singlePlayer : gameserver.players.values())
26             {
27                 if (singlePlayer != null)
28                 {
29                     if (singlePlayer != p)
30                     {
31                         SimplifiedPlayer player = new SimplifiedPlayer();
32                         player.id = p.id;
33                         player.speed = p.speed;
34                         float newSpeed = p.speed * 0.5f;

```

```

35         singlePlayer.speed = newSpeed;
36         this.simplifiedPlayers.put(player.id, player);
37     }
38
39 }
40
41 }
42 this.currentTimer = timer;
43 this.currentCooldown = this.cooldown;
44 }
45 }
46
47 @Override
48 public void onTick(PlayerInfo p)
49 {
50     if (this.currentCooldown > 0)
51     {
52         this.currentCooldown--;
53     }
54     if (this.currentTimer > 1)
55     {
56         this.currentTimer--;
57     }
58     else if (this.currentTimer == 1)
59     {
60         GameServer gameserver = GameServer.getInstance();
61         for(MPPlayer singlePlayer : gameserver.players.values())
62         {
63             if (singlePlayer != null)

```



```
64     {
65         if (singlePlayer != p)
66         {
67             SimplifiedPlayer simplifiedPlayer = this.simplifiedPlayers.
get(singlePlayer.id);
68             if (simplifiedPlayer != null)
69             {
70                 singlePlayer.speed = simplifiedPlayer.speed;
71             }
72         }
73
74     }
75
76 }
77 this.currentTimer --;
78 }
79
80 }
81
82 @Override
83 public String getName()
84 {
85     return this.name;
86 }
87
88 @Override
89 public int getCooldown()
90 {
91     return this.currentCooldown;
```

```
92 }
93 }
```

```
1 package server;
2
3 import shared.SimplifiedPlayer;
4
5 public class DashSkill implements SkillAlgorithm
6 {
7
8     private final int cooldown = 10 * 60; //10 seconds
9     private final String name = "Dash";
10    private int currentCooldown = 0;
11    private final int timer = 1 * 30; //0.25 second
12    private int currentTimer = 0;
13    private SimplifiedPlayer simplified; //For direction
14    private float previousSpeed = 0;
15    @Override
16    public void useSkill(PlayerInfo p)
17    {
18        if (this.currentCooldown == 0)
19        {
20            this.currentTimer = timer;
21            this.currentCooldown = this.cooldown;
22            this.previousSpeed = p.speed;
23            p.speed += 10;
24            this.simplified = new SimplifiedPlayer();
25            this.simplified.isHoldingDown = p.isHoldingDown;
26            this.simplified.isHoldingUp = p.isHoldingUp;
```

```

27     this.simplified.isHoldingLeft = p.isHoldingLeft;
28     this.simplified.isHoldingRight = p.isHoldingRight;
29 }
30 }
31
32 @Override
33 public void onTick(PlayerInfo p)
34 {
35     if (this.currentCooldown > 0)
36     {
37         this.currentCooldown--;
38     }
39
40     if (this.currentTimer > 1)
41     {
42         p.isHoldingDown = this.simplified.isHoldingDown;
43         p.isHoldingUp = this.simplified.isHoldingUp;
44         p.isHoldingLeft = this.simplified.isHoldingLeft;
45         p.isHoldingRight = this.simplified.isHoldingRight;
46         this.currentTimer--;
47     }
48     else if (this.currentTimer == 1)
49     {
50         p.speed = this.previousSpeed;
51         this.currentTimer--;
52     }
53
54 }
55

```

```
56 @Override
57 public String getName()
58 {
59     return this.name;
60 }
61
62 @Override
63 public int getCooldown()
64 {
65     return this.currentCooldown;
66 }
67 }
```

```
1 package server;
2
3
4 public class TeleportSkill implements SkillAlgorithm
5 {
6
7     private final int cooldown = 10 * 60; //10 seconds
8     private final String name = "Teleport";
9     private int currentCooldown = 0;
10
11     @Override
12     public void useSkill(PlayerInfo p)
13     {
14         if (this.currentCooldown == 0)
15         {
16             GameServer gameserver = GameServer.getInstance();
```

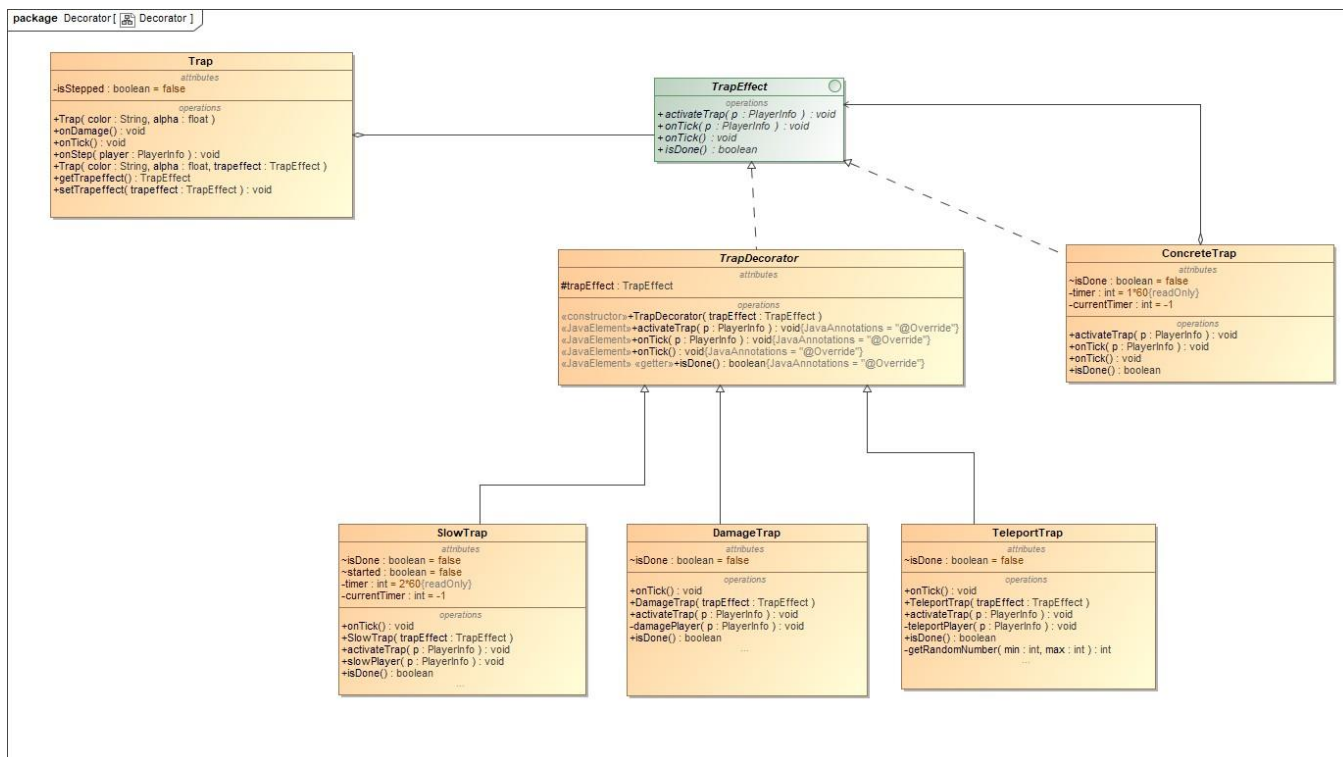
```

17     boolean teleported = false;
18     int maxRetry = 60;
19     int retry = 0;
20     while (!teleported && retry < maxRetry)
21     {
22         int randomCoordX = getRandomNumber (0, gameserver.gameBoard.gridSize);
23         int randomCoordY = getRandomNumber (0, gameserver.gameBoard.gridSize);
24         if (gameserver.gameBoard.objects[randomCoordX][randomCoordY] instanceof Ground)
25         {
26             p.coordinate.x = randomCoordX * (gameserver.gameBoard.size / gameserver.gameBoard.gridSize);
27             p.coordinate.y = randomCoordY * (gameserver.gameBoard.size / gameserver.gameBoard.gridSize);
28             teleported = true;
29         }
30         retry++;
31     }
32     this.currentCooldown = this.cooldown;
33 }
34
35
36 @Override
37 public void onTick(PlayerInfo p)
38 {
39     if (this.currentCooldown > 0)
40     {

```

```
41     this.currentCooldown --;
42 }
43 }
44
45 @Override
46 public String getName()
47 {
48     return this.name;
49 }
50
51 @Override
52 public int getCooldown()
53 {
54     return this.currentCooldown;
55 }
56
57 private int getRandomNumber(int min, int max)
58 {
59     return (int) ((Math.random() * (max - min)) + min);
60 }
61
62 }
```

5.1.3. Decorator



5.6 pav. Decorator diagrama

Decorator šablono poreikis atsirado norint turėti skirtingus spąstus, su skirtingais efektais, kurie atliekami žaidėjui. Šie efektai gali sluoksniuotis.

```

1 package server;
2
3 public class Trap extends GameObject{
4
5
6     private TrapEffect trapEffect;
7     private boolean isStepped = false;
8
9     public Trap(String color, float alpha, TrapEffect trapeffect)
10    {
  
```

```
11         super (color , alpha );
12         this .trapeffect = trapeffect;
13     }
14
15     public void onDamage()
16     {
17         if ( this .isDestroyable)
18             isDead = true ;
19     }
20
21     public void onTick ()
22     {
23         this .trapeffect .onTick ();
24         if ( this .trapeffect .isDone ())
25         {
26             isDead = true ;
27         }
28     }
29
30     public void onStep (PlayerInfo player)
31     {
32         if (!isStepped )
33         {
34             this .trapeffect .activateTrap (player);
35             isStepped = true ;
36         }
37     }
38
39     public TrapEffect getTrapeffect ()
40     {
```



```
40     return trapeffect;
41 }
42
43 public void setTrapeffect(TrapEffect trapeffect)
44 {
45     this.trapeffect = trapeffect;
46 }
47
48 }
```

```
1 package server;
2
3 public interface TrapEffect
4 {
5     void activateTrap (PlayerInfo p);
6     void onTick ();
7     boolean isDone ();
8 }
```

```
1 package server;
2
3 public class ConcreteTrap implements TrapEffect
4 {
5     private PlayerInfo p;
6     boolean isDone = false;
7     private final int timer = 1 * 60; //second
8     private int currentTimer = -1;
9     @Override
10    public void activateTrap (PlayerInfo p)
11    {
```

```

12     this.p = p;
13     this.p.speed = 0;
14     GameServer gameserver = GameServer.getInstance();
15     gameserver.players.get(this.p.id).speed = this.p.speed;
16     this.currentTimer = timer;
17 }
18
19 @Override
20 public void onTick()
21 {
22
23     if (this.currentTimer > 1)
24     {
25         this.currentTimer--;
26     }
27     else if (this.currentTimer == 1)
28     {
29         GameServer gameserver = GameServer.getInstance();
30         gameserver.players.get(this.p.id).speed = this.p.baseSpeed;
31         this.currentTimer--;
32     }
33     else if (this.currentTimer == 0)
34     {
35         this.isDone = true;
36     }
37
38 }
39
40 @Override

```

```
41 public boolean isDone ()
42 {
43     return isDone ;
44 }
45
46 }
```

```
1 package server;
2
3 public abstract class TrapDecorator implements TrapEffect
4 {
5     protected TrapEffect trapEffect;
6
7     public TrapDecorator (TrapEffect trapEffect)
8     {
9         this.trapEffect = trapEffect;
10    }
11
12    @Override
13    public void activateTrap (PlayerInfo p)
14    {
15        this.trapEffect.activateTrap (p);
16    }
17
18    @Override
19    public void onTick ()
20    {
21        this.trapEffect.onTick ();
22    }
```

```

23
24 @Override
25 public boolean isDone ()
26 {
27     return this.trapEffect.isDone ();
28 }
29
30 }

```

```

1 package server;
2
3 public class SlowTrap extends TrapDecorator
4 {
5
6     boolean isDone = false;
7     boolean started = false;
8     private PlayerInfo p;
9     private final int timer = 2 * 60; //4 seconds
10    private int currentTimer = -1;
11    public SlowTrap (TrapEffect trapEffect)
12    {
13        super (trapEffect);
14    }
15
16    public void activateTrap (PlayerInfo p)
17    {
18        super . activateTrap (p);
19        this . p = p;
20    }

```

```

21
22 public void onTick ()
23 {
24     super.onTick ();
25     slowPlayer (p);
26 }
27
28 public void slowPlayer (PlayerInfo p)
29 {
30     if (super.isDone () && !started)
31     {
32         this.p.speed = 0;
33         GameServer gameserver = GameServer.getInstance ();
34         gameserver.players.get (this.p.id).speed = this.p.baseSpeed * 0.5
35 f;
36         this.currentTimer = timer ;
37         started = true ;
38     }
39
40     if (this.currentTimer > 1)
41     {
42         this.currentTimer --;
43     }
44     else if (this.currentTimer == 1)
45     {
46         GameServer gameserver = GameServer.getInstance ();
47         gameserver.players.get (this.p.id).speed = this.p.baseSpeed ;
48         this.currentTimer --;
49     }

```

```
49     else if (this.currentTimer == 0)
50     {
51         this.isDone = true;
52     }
53 }
54 public boolean isDone ()
55 {
56     return super.isDone () && isDone;
57 }
58 }
```

```
1 package server;
2
3 public class DamageTrap extends TrapDecorator
4 {
5
6     boolean isDone = false;
7     public DamageTrap(TrapEffect trapEffect)
8     {
9         super(trapEffect);
10    }
11
12    public void activateTrap (PlayerInfo p)
13    {
14        super.activateTrap (p);
15        damagePlayer (p);
16    }
17
18    private void damagePlayer (PlayerInfo p)
```

```

19 {
20     GameServer gameserver = GameServer.getInstance();
21     gameserver.players.get(p.id).health--;
22     isDone = true;
23 }
24
25 public void onTick()
26 {
27     super.onTick();
28 }
29
30 @Override
31 public boolean isDone()
32 {
33     return super.isDone() && isDone;
34 }
35 }

```

```

1 package server;
2
3 public class TeleportTrap extends TrapDecorator
4 {
5
6     boolean isDone = false;
7     public TeleportTrap(TrapEffect trapEffect)
8     {
9         super(trapEffect);
10    }
11

```

```

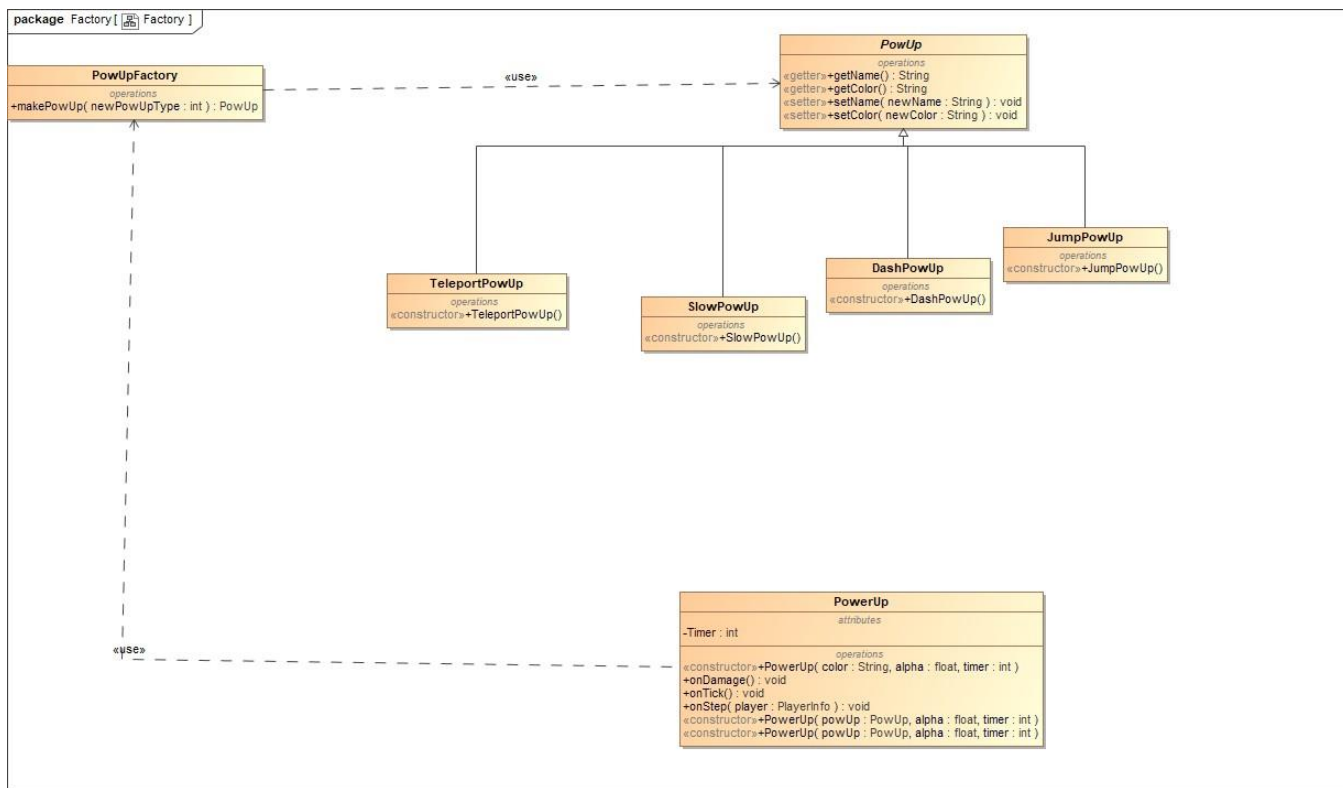
12 public void activateTrap (PlayerInfo p)
13 {
14     super . activateTrap (p);
15     teleportPlayer (p);
16
17 }
18
19 private void teleportPlayer (PlayerInfo p)
20 {
21     GameServer gameserver = GameServer . getInstance ();
22     boolean teleported = false;
23     int maxRetry = 60;
24     int retry = 0;
25     while (!teleported && retry < maxRetry)
26     {
27         int randomCoordX = getRandomNumber (0, gameserver . gameBoard .
gridSize);
28         int randomCoordY = getRandomNumber (0, gameserver . gameBoard .
gridSize);
29         if ( gameserver . gameBoard . objects [ randomCoordX ] [ randomCoordY ]
instanceof Ground)
30         {
31             p . coordinate . x = randomCoordX * ( gameserver . gameBoard . size /
gameserver . gameBoard . gridSize);
32             p . coordinate . y = randomCoordY * ( gameserver . gameBoard . size /
gameserver . gameBoard . gridSize);
33             teleported = true;
34         }
35         retry ++;

```



```
36     }
37     isDone = true ;
38 }
39
40 public void onTick ()
41 {
42     super . onTick () ;
43 }
44
45 @Override
46 public boolean isDone ()
47 {
48     return super . isDone () && isDone ;
49 }
50
51 private int getRandomNumber (int min , int max)
52 {
53     return (int) ((Math.random () * (max - min)) + min) ;
54 }
55 }
```

5.1.4. Abstract Factory



5.7 pav. Abstract Factory diagrama

Abstract Factory šablonas padeda lanksčiai sukurti skirtingus žaidimo objektų rinkinius skirtingiems lygiams.

```

1 package server;
2
3 public abstract class AbstractFactory
4 {
5     protected BombObserver bombObserver;
6     public abstract GameObject createTrap();
7     public abstract GameObject createWall(boolean destroyable);
8     public abstract GameObject createBomb(int ownerid);
9     public abstract GameObject createPowerUp();
10    public abstract GameObject createGround();
  
```

```

11
12     public void SetBombObserver(BombObserver observer)
13     {
14         this.bombObserver = observer;
15     }
16 }

```

```

1 package server;
2
3
4 import java.util.Hashtable;
5 import java.util.Random;
6
7 public class Stage1Factory extends AbstractFactory {
8
9     private Hashtable<String, String> colors;
10
11     public Stage1Factory(){
12         this.colors = new Hashtable<>();
13         this.colors.put("Wall", "#2e2203");
14         this.colors.put("DesWall", "#8c8674");
15         this.colors.put("Ground", "#348C31");
16         this.colors.put("Bomb", "#6A2E35");
17         this.colors.put("PowerUp", "#BAD094");
18         this.colors.put("Trap", "#373D20");
19     }
20
21     @Override
22     public GameObject createTrap()

```

```

23     {
24         return new Trap(this.colors.get("Trap"), 1, new ConcreteTrap());
25     };
26
27     @Override
28     public GameObject createWall(boolean destroyable)
29     {
30         String wallType = "Wall";
31
32         if(destroyable)
33             wallType = "DesWall";
34
35         return new Wall(this.colors.get(wallType), 1, destroyable);
36     }
37
38     @Override
39     public GameObject createBomb(int ownerid) {
40         return new Bomb(colors.get("Bomb"), 1, 90, ownerid, this.
bombObserver);
41     }
42
43     @Override
44     public GameObject createPowerUp() {
45
46         PowUpFactory powFactory = new PowUpFactory();
47         PowUp powUp = null;
48
49         Random rand = new Random();

```

```

50         int randomNum = rand.nextInt((4 - 1) + 1) + 1;
51
52         powUp = powFactory.makePowUp(randomNum);
53
54
55         return new PowerUp(powUp, 1, 5);
56     }
57
58     @Override
59     public GameObject createGround()
60     {
61         return new Ground(colors.get("Ground"), 1);
62     }
63
64 }

```

```

1 package server;
2
3
4 import java.util.Hashtable;
5 import java.util.Random;
6
7 public class Stage2Factory extends AbstractFactory {
8
9     private Hashtable<String, String> colors;
10
11     public Stage2Factory() {
12         this.colors = new Hashtable<>();
13         this.colors.put("Wall", "#ECE5F0");

```

```

14         this.colors.put("DesWall", "#CEB5A7");
15         this.colors.put("Ground", "#FFA3AF");
16         this.colors.put("Bomb", "#210124");
17         this.colors.put("PowerUp", "#87FF65");
18         this.colors.put("Trap", "#8EA604");
19     }
20
21     @Override
22     public GameObject createTrap ()
23     {
24         return new Trap(this.colors.get("Trap"), 1, new ConcreteTrap())
25     ;
26     }
27
28     @Override
29     public GameObject createWall(boolean destroyable)
30     {
31         String wallType = "Wall";
32
33         if(destroyable)
34             wallType = "DesWall";
35
36         return new Wall(this.colors.get(wallType), 1, destroyable);
37     }
38
39     @Override
40     public GameObject createBomb(int ownerid) {
41         return new Bomb(colors.get("Bomb"), 1, 90, ownerid, this.
42         bombObserver);

```

```

41     }
42
43     @Override
44     public GameObject createPowerUp () {
45
46         PowUpFactory powFactory = new PowUpFactory ();
47         PowUp powUp = null;
48
49         Random rand = new Random();
50         int randomNum = rand.nextInt((4 - 1) + 1) + 1;
51
52         powUp = powFactory.makePowUp(randomNum);
53
54
55         return new PowerUp(powUp, 1, 5);
56     }
57
58     @Override
59     public GameObject createGround ()
60     {
61         return new Ground(colors.get("Ground"), 1);
62     }
63
64 }

```

```

1 package server;
2
3 public abstract class TrapDecorator implements TrapEffect
4 {

```

```
5  protected TrapEffect trapEffect;
6
7  public TrapDecorator (TrapEffect trapEffect)
8  {
9      this.trapEffect = trapEffect;
10 }
11
12 @Override
13 public void activateTrap (PlayerInfo p)
14 {
15     this.trapEffect.activateTrap (p);
16 }
17
18 @Override
19 public void onTick ()
20 {
21     this.trapEffect.onTick ();
22 }
23
24 @Override
25 public boolean isDone ()
26 {
27     return this.trapEffect.isDone ();
28 }
29
30 }
```

```
1 package server;
2 import java.util.Random;
```



```

3
4 public class PowerUp extends GameObject
5 {
6     private int Timer ;
7     private PowUp powUp;
8
9     public PowerUp(PowUp powUp, float alpha , int timer ){
10
11
12         super(powUp.getColor () , alpha );
13         this.powUp = powUp;
14         this.Timer = timer ;
15         this.isWalkable = true ;
16
17
18
19     }
20
21     public void onDamage(){
22         if (this.isDestroyable)
23             System.out.println("Wall has been destroyed");
24     }
25     public void onTick (){
26         //         this.Timer --;
27         //         if (this.Timer <= 0)
28         //             isDead = true ;
29     }
30
31

```

```
32
33
34
35
36
37
38 public void onStep(PlayerInfo player){
39     if (this.isWalkable )
40     {
41         if (!isDead )
42         {
43             GameServer gameServer = GameServer.getInstance();
44
45
46             if (powUp.getName().equals("Jump"))
47                 gameServer.players.get(player.id).setSkillAlgorithm(new
JumpSkill());
48
49             else if (powUp.getName().equals("Dash"))
50                 gameServer.players.get(player.id).setSkillAlgorithm(new
DashSkill());
51
52             else if (powUp.getName().equals("Slow"))
53                 gameServer.players.get(player.id).setSkillAlgorithm(new
SlowAllPlayersSkill());
54
55             else if (powUp.getName().equals("Teleport"))
56                 gameServer.players.get(player.id).setSkillAlgorithm(new
TeleportSkill());
```

```
57
58
59
60
61         isDead = true ;
62     }
63 }
64     System.out.println("You gain powerup!");
65 }
66 }
```

```
1 package server;
2
3 public class Ground extends GameObject{
4
5     public Ground(String color , float alpha){
6         super(color , alpha ) ;
7         this.isWalkable = true ;
8     }
9
10    @Override
11    public void onDamage() {
12
13    }
14
15    @Override
16    public void onTick () {
17
18    }
```

```

19
20     @Override
21     public void onStep(PlayerInfo player) {
22
23     }
24 }

1 package server;
2
3 import java.util.List;
4 import java.util.Stack;
5
6 public class Bomb extends GameObject implements BombObservable{
7
8     private int Timer;
9     private int OwnerId;
10    private int ExplosionRadius = 3;
11    private List<BombObserver> observers = new Stack<BombObserver>();
12
13    public Bomb(String color, float alpha, int timer, int ownerid,
BombObserver observer){
14        super(color, alpha);
15        this.Timer = timer;
16        this.OwnerId = ownerid;
17        this.add(observer);
18        this.isWalkable = true;
19    }
20
21    public void onDamage(){

```

```
22         //cannot be destroyed?
23     }
24     public void onTick () {
25         this.Timer - -;
26         if (this.Timer <= 0)
27             notifyObservers ();
28     }
29
30     public void onStep (PlayerInfo player) {
31         //cannot be stepped on?
32     }
33
34     public void add (BombObserver observer)
35     {
36         this.observers.add (observer);
37     }
38
39     public void remove (BombObserver observer)
40     {
41         this.observers.remove (observer);
42     }
43
44     public void notifyObservers ()
45     {
46         for (BombObserver observer :
47             observers) {
48             observer.explode (this);
49
50         }
```

```
51     }
52
53     public int explosionRadius () {
54         if (this.Timer <= 0) return this.ExpllosionRadius ;
55         else return 0;
56     }
57 }
```

```
1 package server;
2
3 public class Wall extends GameObject{
4
5
6     public Wall(String color , float alpha, boolean destroyable){
7         super(color , alpha);
8         this.isDestroyable = destroyable;
9     }
10
11     public void onDamage()
12     {
13         if(this.isDestroyable)
14             isDead = true ;
15     }
16
17     public void onTick()
18     {
19
20     }
21 }
```

```

22     public void onStep(PlayerInfo player)
23     {
24         if (!this.isWalkable)
25             System.out.println("Can't walk on this!");
26     }
27 }

```

```

1 package server;
2
3 public class Trap extends GameObject{
4
5
6     private TrapEffect trapeffect;
7     private boolean isStepped = false;
8
9     public Trap(String color, float alpha, TrapEffect trapeffect)
10    {
11        super(color, alpha);
12        this.trapeffect = trapeffect;
13    }
14
15    public void onDamage()
16    {
17        if (this.isDestroyable)
18            isDead = true;
19    }
20    public void onTick()
21    {
22        this.trapeffect.onTick();

```

```
23     if (this.trapeffect.isDone())
24     {
25         isDead = true ;
26     }
27 }
28
29 public void onStep(PlayerInfo player)
30 {
31     if (!isStepped)
32     {
33         this.trapeffect.activateTrap(player);
34         isStepped = true ;
35     }
36 }
37
38 public TrapEffect getTrapeffect()
39 {
40     return trapeffect;
41 }
42
43 public void setTrapeffect(TrapEffect trapeffect)
44 {
45     this.trapeffect = trapeffect;
46 }
47
48 }
```

```
1 package server;
```

```
2
```



```
3 public abstract class GameObject {
4     public boolean isWalkable ;
5     public boolean isDestroyable ;
6     public boolean isDead ;
7
8     GameObjectDelegate gameobjectdelegate ;
9
10    public String color ;
11    public float alpha ;
12
13    public GameObject(String color , float alpha) {
14        super () ;
15        this.alpha = alpha ;
16        this.color = color ;
17    }
18
19    public void sayHello () {
20        System.out.println ("GameObject") ;
21    }
22
23    public void setDestroyable(boolean option){
24        this.isDestroyable = option ;
25    }
26
27    public void setWalkable (boolean option){
28        this.isWalkable = option ;
29    }
30
31    public abstract void onDamage() ;
```

```
32     public abstract void onTick();
33     public abstract void onStep(PlayerInfo player);
34
35 }
```

```
1 package server;
2
3 import java.util.ArrayList;
4 import java.util.Random;
5
6 import shared.*;
7
8 public class GameBoard implements Cloneable{
9
10     public final int size = 1000;
11     public final int gridSize = 20;
12     public GameObject[][] objects;
13     private BombObserver bombObserver;
14     private AbstractFactory factory;
15
16     private int currentTick = 0;
17     private int powerUpCounter = 0;
18     private int timeToCreatePowerUp = 10;
19
20     private IStageBuilder stage1builder;
21     private IStageBuilder stage2builder;
22     private IStageBuilder stage3builder;
23
24     public GameBoard( AbstractFactory factory )
```

```

25 {
26     this.factory = factory;
27     this.objects = new GameObject[ gridSize ][ gridSize ];
28     this.stage1builder = new Stage1Builder( gridSize );
29     this.stage2builder = new Stage2Builder( gridSize );
30     this.stage3builder = new Stage3Builder( gridSize );
31
32     StageDirector stageDirector = new StageDirector( stage3builder );
33
34     stageDirector.makeStage();
35
36     Stage stage1 = stageDirector.getStage();
37
38     for( Coordinates kor : stage1.getGrounds() )
39         this.objects[ kor.getX() ][ kor.getY() ] = this.factory.
createGround();
40
41     int kiekis = 76;
42     boolean sunaikinama = false;
43
44     for( Coordinates kor : stage1.getWalls() ){
45
46         this.objects[ kor.getX() ][ kor.getY() ] = this.factory.
createWall( sunaikinama );
47         kiekis -= 1;
48         if( kiekis == 0 ) sunaikinama = true;
49     }
50
51

```

```
52
53     bombObserver = new BombObserver ( this );
54
55
56     this . factory . SetBombObserver ( bombObserver );
57
58
59
60
61
62     //test
63
64     //TODO wrong but ok for now
65     this . objects [ 17 ][ 3 ] = this . factory . createTrap ( ) ;
66
67     this . objects [ 15 ][ 3 ] = this . factory . createTrap ( ) ;
68     Trap modified = (Trap) this . objects [ 15 ][ 3 ];
69     modified . setTrapeffect ( new DamageTrap ( new ConcreteTrap ( ) ) );
70     this . objects [ 15 ][ 3 ] = modified ;
71
72     this . objects [ 13 ][ 3 ] = this . factory . createTrap ( ) ;
73     Trap modified1 = (Trap) this . objects [ 13 ][ 3 ];
74     modified1 . setTrapeffect ( new TeleportTrap ( new ConcreteTrap ( ) ) );
75     this . objects [ 13 ][ 3 ] = modified1 ;
76
77     this . objects [ 11 ][ 3 ] = this . factory . createTrap ( ) ;
78     Trap modified2 = (Trap) this . objects [ 11 ][ 3 ];
79     modified2 . setTrapeffect ( new SlowTrap ( new ConcreteTrap ( ) ) );
80     this . objects [ 11 ][ 3 ] = modified2 ;
```

```

81
82     this.objects[9][3] = this.factory.createTrap();
83     Trap modified3 = (Trap) this.objects[9][3];
84     modified3.setTrapeffect(new DamageTrap(new TeleportTrap(new
SlowTrap(new ConcreteTrap()))));
85     this.objects[9][3] = modified3;
86
87 }
88
89 public void paleistiKopija(){
90     GameBoard copy = copyDeep();
91
92
93 }
94
95
96
97 public GameBoard copyDeep(){
98
99     try {
100         return (GameBoard) this.clone();
101     } catch (CloneNotSupportedException e) {
102         e.printStackTrace();
103         return null;
104     }
105 }
106
107 public void SpawnBomb( PlayerInfo player)
108 {

```

```

109         int x = Math.round(player.coordinate.x/(size/gridSize));
110         int y = Math.round(player.coordinate.y/(size/gridSize));
111         this.objects[x][y] = this.factory.createBomb(player.id);
112
113     }
114
115     public void ClearTarget(int x, int y){
116         this.objects[x][y] = this.factory.createGround();
117         System.out.println("Removing bomb from location " + x + " " + y
118     );
119     }
120
121     public SimplifiedGameBoard getSimpleGameBoard()
122     {
123         SimplifiedGameBoard simpleGameboard = new SimplifiedGameBoard(this.
124 size, this.gridSize);
125
126         for (int i = 0; i < this.gridSize; i++)
127         {
128             for (int j = 0; j < this.gridSize; j++)
129             {
130                 simpleGameboard.objects[i][j] = new SimplifiedGameObject
131 (this.objects[i][j].color, ObjectType.GROUND);
132             }
133         }
134         return simpleGameboard;
135     }
136
137     private void spawnPowerUp(){

```

```
135 Random rand = new Random();
136
137 if (powerUpCounter > 4)
138     return;
139
140 while (true){
141     int x = rand.nextInt((19 - 0) + 1) + 0;
142     int y = rand.nextInt((19 - 0) + 1) + 0;
143
144
145     if (this.objects[x][y] instanceof Ground){
146         this.objects[x][y] = this.factory.createPowerUp();
147         powerUpCounter+=1;
148         break;
149     }
150
151 }
152
153
154
155
156
157
158
159
160 }
161
162
163 public void runTick()
```

```

164 {
165     currentTick +=1;
166
167     if(currentTick % (60 * timeToCreatePowerUp)== 0) // kas 10s
sukurti nauja powerUp
168         spawnPowerUp () ;
169
170     for(int x = 0; x < this.gridSize; x++)
171     {
172         for (int y = 0; y < this.gridSize; y++)
173         {
174             if (this.objects[x][y].isDead) {
175
176
177
178                 if (this.objects[x][y] instanceof PowerUp)
179                 {
180                     System.out.println("Sunaikinau powerUpa");
181                     powerUpCounter -=1;
182                 }
183
184                 this.ClearTarget(x, y);
185
186
187
188             }
189
190             objects[x][y].onTick();
191         }

```

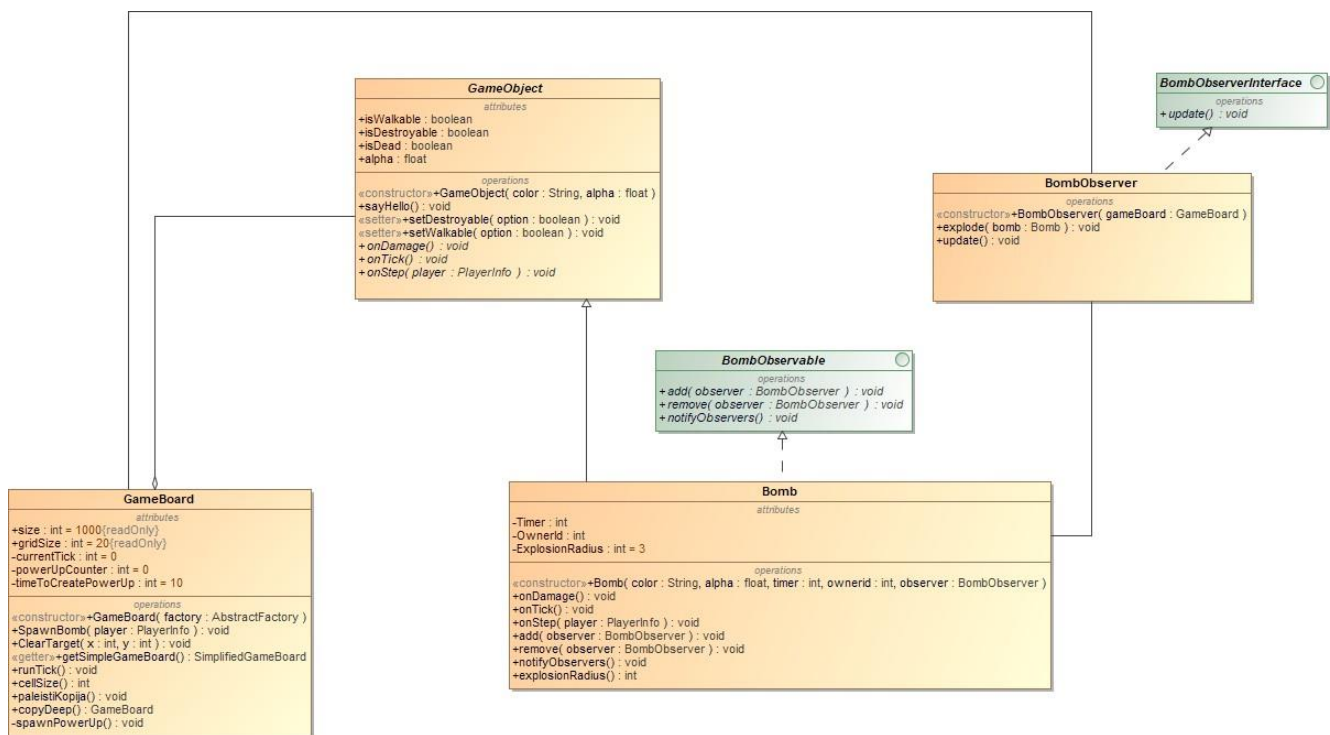


```

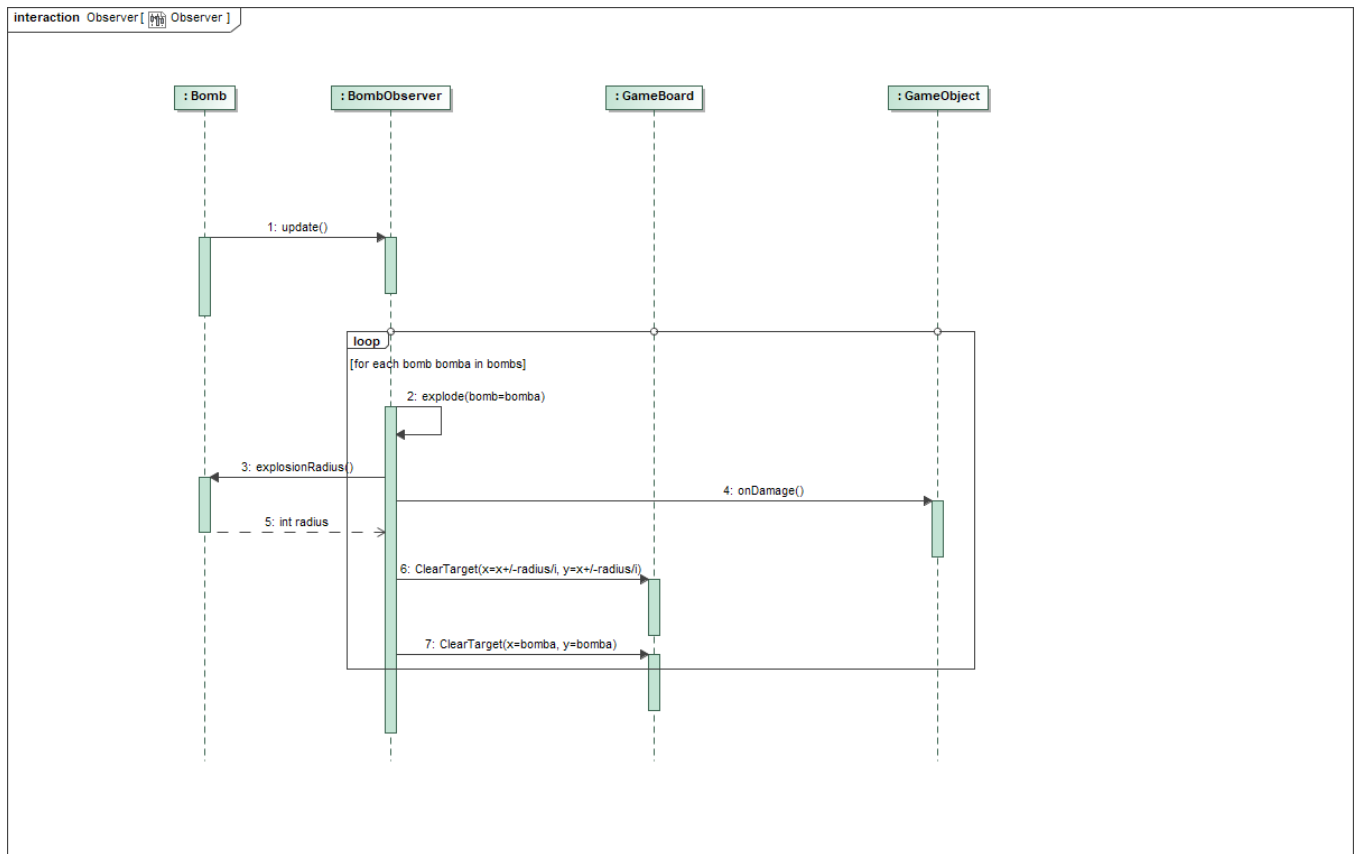
192     }
193 }
194
195 public int cellSize()
196 {
197     return size / gridSize;
198 }
199 }

```

5.1.5. Observer



5.8 pav. Observer diagrama



5.9 pav. Observer usecase

Observer optimaliai apdoroja bombos informacija tik tada, kai bomba yra pasiruošusi sprogti.

```

1 package server;
2
3 import java.util.List;
4 import java.util.Stack;
5
6 public class Bomb extends GameObject implements BombObservable{
7
8     private int Timer ;
9     private int OwnerId ;
10    private int ExplosionRadius = 3;
11    private List<BombObserver> observers = new Stack<BombObserver>();
12

```

```
13     public Bomb(String color , float alpha , int timer , int ownerid ,
BombObserver observer){
14         super (color , alpha );
15         this.Timer = timer ;
16         this.OwnerId = ownerid ;
17         this.add(observer);
18         this.isWalkable = true ;
19     }
20
21     public void onDamage(){
22         //cannot be destroyed?
23     }
24     public void onTick (){
25         this.Timer - -;
26         if ( this.Timer <= 0)
27             notifyObservers ();
28     }
29
30     public void onStep (PlayerInfo player){
31         //cannot be stepped on?
32     }
33
34     public void add (BombObserver observer)
35     {
36         this.observers.add(observer);
37     }
38
39     public void remove (BombObserver observer)
40     {
```

```

41         this.observers.remove(observer);
42     }
43
44     public void notifyObservers()
45     {
46         for (BombObserver observer:
47             observers){
48             observer.explode(this);
49
50         }
51     }
52
53     public int explosionRadius() {
54         if (this.Timer <= 0) return this.ExpllosionRadius;
55         else return 0;
56     }
57 }

```

```

1 package server;
2
3 public interface BombObservable {
4     public void add (BombObserver observer);
5     public void remove (BombObserver observer);
6     public void notifyObservers ();
7
8 }

```

```

1 package server;
2
3 import java.util.List;

```

```

4 import java.util.Stack;
5
6 public class BombObserver implements BombObserverInterface {
7     private GameBoard gameBoard;
8     private List<Bomb> bombs = new Stack<Bomb>();
9
10
11     public BombObserver(GameBoard gameBoard){
12         this.gameBoard = gameBoard;
13     }
14
15     public void update() {
16         for(int x = 0; x < this.gameBoard.gridSize; x++) {
17             for (int y = 0; y < this.gameBoard.gridSize; y++) {
18                 if (this.gameBoard.objects[x][y].getClass().equals(Bomb
19                     .class))
20                     {
21                         bombs.add((Bomb) this.gameBoard.objects[x][y]);
22                     }
23             }
24
25             for (Bomb bomb: bombs) {
26                 if (bomb.explosionRadius() > 0){
27                     explode(bomb);
28                 }
29             }
30         }
31

```

```

32 public void explode (Bomb bomb){
33     for(int x = 0; x < this.gameBoard.gridSize; x++)
34     {
35         for (int y = 0; y < this.gameBoard.gridSize; y++)
36         {
37             if (this.gameBoard.objects[x][y] == bomb)
38             {
39                 for(int i = 0; i < bomb.explosionRadius(); i++) {
40                     //TODO: Make a sophisticated explosion radius
41                     calculation
42                     this.gameBoard.ClearTarget(x, y);
43                     if(x + i < this.gameBoard.gridSize) {
44                         this.gameBoard.objects[x + i][y].onDamage()
45                     ;
46                     if (this.gameBoard.objects[x + i][y].isDead
47                     ) this.gameBoard.ClearTarget(x + 1, y);
48                     }
49                     if(y + i < this.gameBoard.gridSize) {
50                         this.gameBoard.objects[x][y + i].onDamage()
51                     ;
52                     if (this.gameBoard.objects[x][y + i].isDead
53                     ) this.gameBoard.ClearTarget(x, y + 1);
54                     }
55                     if(x - i > 0) {
56                         this.gameBoard.objects[x - i][y].onDamage()
57                     ;
58                     if (this.gameBoard.objects[x - i][y].isDead
59                     ) this.gameBoard.ClearTarget(x - 1, y);
60                     }
61                 }
62             }
63         }
64     }
65 }

```

```

54         if (y - i > 0) {
55             this.gameBoard.objects[x][y - i].onDamage()
56             ;
57             if (this.gameBoard.objects[x][y - i].isDead
58             ) this.gameBoard.ClearTarget(x, y - 1);
59         }
60     }
61 }
62 }
63 }

```

```

1 package server;
2
3 public interface BombObserverInterface {
4     public void update();
5 }

```

```

1 package server;
2
3 public abstract class GameObject {
4     public boolean isWalkable;
5     public boolean isDestroyable;
6     public boolean isDead;
7
8     GameObjectDelegate gameobjectdelegate;
9
10    public String color;
11    public float alpha;

```

```

12
13     public GameObject(String color , float alpha) {
14         super();
15         this.alpha = alpha;
16         this.color = color;
17     }
18
19     public void sayHello () {
20         System.out.println("GameObject");
21     }
22
23     public void setDestroyable(boolean option){
24         this.isDestroyable = option;
25     }
26
27     public void setWalkable(boolean option){
28         this.isWalkable = option;
29     }
30
31     public abstract void onDamage();
32     public abstract void onTick();
33     public abstract void onStep(PlayerInfo player);
34
35 }

```

```

1 package server;
2
3 import java.util.ArrayList;
4 import java.util.Random;

```



```
5
6 import shared.*;
7
8 public class GameBoard implements Cloneable{
9
10     public final int size = 1000;
11     public final int gridSize = 20;
12     public GameObject[][] objects;
13     private BombObserver bombObserver;
14     private AbstractFactory factory;
15
16     private int currentTick = 0;
17     private int powerUpCounter = 0;
18     private int timeToCreatePowerUp = 10;
19
20     private IStageBuilder stage1builder;
21     private IStageBuilder stage2builder;
22     private IStageBuilder stage3builder;
23
24     public GameBoard( AbstractFactory factory )
25     {
26         this.factory = factory;
27         this.objects = new GameObject[ gridSize ][ gridSize ];
28         this.stage1builder = new Stage1Builder( gridSize );
29         this.stage2builder = new Stage2Builder( gridSize );
30         this.stage3builder = new Stage3Builder( gridSize );
31
32         StageDirector stageDirector = new StageDirector( stage3builder );
33
```

```

34     stageDirector.makeStage();
35
36     Stage stage1 = stageDirector.getStage();
37
38     for(Coordinates kor : stage1.getGrounds())
39         this.objects[kor.getX()][kor.getY()] = this.factory.
createGround();
40
41     int kiekis = 76;
42     boolean sunaikinama = false;
43
44     for(Coordinates kor : stage1.getWalls()){
45
46         this.objects[kor.getX()][kor.getY()] = this.factory.
createWall(sunaikinama);
47         kiekis -=1;
48         if(kiekis == 0) sunaikinama = true;
49     }
50
51
52
53     bombObserver = new BombObserver(this);
54
55
56     this.factory.SetBombObserver(bombObserver);
57
58
59
60

```

```

61
62 //test
63
64 //TODO wrong but ok for now
65 this.objects[17][3] = this.factory.createTrap();
66
67 this.objects[15][3] = this.factory.createTrap();
68 Trap modified = (Trap) this.objects[15][3];
69 modified.setTrapeffect(new DamageTrap (new ConcreteTrap()));
70 this.objects[15][3] = modified;
71
72 this.objects[13][3] = this.factory.createTrap();
73 Trap modified1 = (Trap) this.objects[13][3];
74 modified1.setTrapeffect(new TeleportTrap (new ConcreteTrap()));
75 this.objects[13][3] = modified1;
76
77 this.objects[11][3] = this.factory.createTrap();
78 Trap modified2 = (Trap) this.objects[11][3];
79 modified2.setTrapeffect(new SlowTrap (new ConcreteTrap()));
80 this.objects[11][3] = modified2;
81
82 this.objects[9][3] = this.factory.createTrap();
83 Trap modified3 = (Trap) this.objects[9][3];
84 modified3.setTrapeffect(new DamageTrap (new TeleportTrap (new
SlowTrap (new ConcreteTrap()))));
85 this.objects[9][3] = modified3;
86
87 }
88

```

```

89     public void paleistiKopija(){
90         GameBoard copy = copyDeep();
91
92
93     }
94
95
96
97     public GameBoard copyDeep(){
98
99         try {
100             return (GameBoard) this.clone();
101         } catch (CloneNotSupportedException e) {
102             e.printStackTrace();
103             return null;
104         }
105     }
106
107     public void SpawnBomb(PlayerInfo player)
108     {
109         int x = Math.round(player.coordinate.x/(size/gridSize));
110         int y = Math.round(player.coordinate.y/(size/gridSize));
111         this.objects[x][y] = this.factory.createBomb(player.id);
112
113     }
114
115     public void ClearTarget(int x, int y){
116         this.objects[x][y] = this.factory.createGround();
117         System.out.println("Removing bomb from location " + x + " " + y

```

```

117 );
118 }
119
120 public SimplifiedGameBoard getSimpleGameBoard ()
121 {
122     SimplifiedGameBoard simpleGameboard = new SimplifiedGameBoard ( this .
size , this . gridSize );
123
124     for ( int i = 0; i < this . gridSize; i ++ )
125     {
126         for ( int j = 0; j < this . gridSize; j ++ )
127         {
128             simpleGameboard . objects [ i ][ j ] = new SimplifiedGameObject
( this . objects [ i ][ j ] . color , ObjectType . GROUND );
129         }
130     }
131     return simpleGameboard ;
132 }
133
134 private void spawnPowerUp () {
135     Random rand = new Random ();
136
137     if ( powerUpCounter > 4 )
138         return ;
139
140     while ( true ) {
141         int x = rand . nextInt ( ( 19 - 0 ) + 1 ) + 0 ;
142         int y = rand . nextInt ( ( 19 - 0 ) + 1 ) + 0 ;
143

```

```

144
145         if (this.objects[x][y] instanceof Ground){
146             this.objects[x][y] = this.factory.createPowerUp();
147             powerUpCounter+=1;
148             break;
149         }
150
151     }
152
153
154
155
156
157
158
159
160 }
161
162
163 public void runTick()
164 {
165     currentTick +=1;
166
167     if (currentTick % (60 * timeToCreatePowerUp) == 0) // kas 10s
168         sukurti nauja powerUp
169         spawnPowerUp();
170
171     for (int x = 0; x < this.gridSize; x++)
172     {

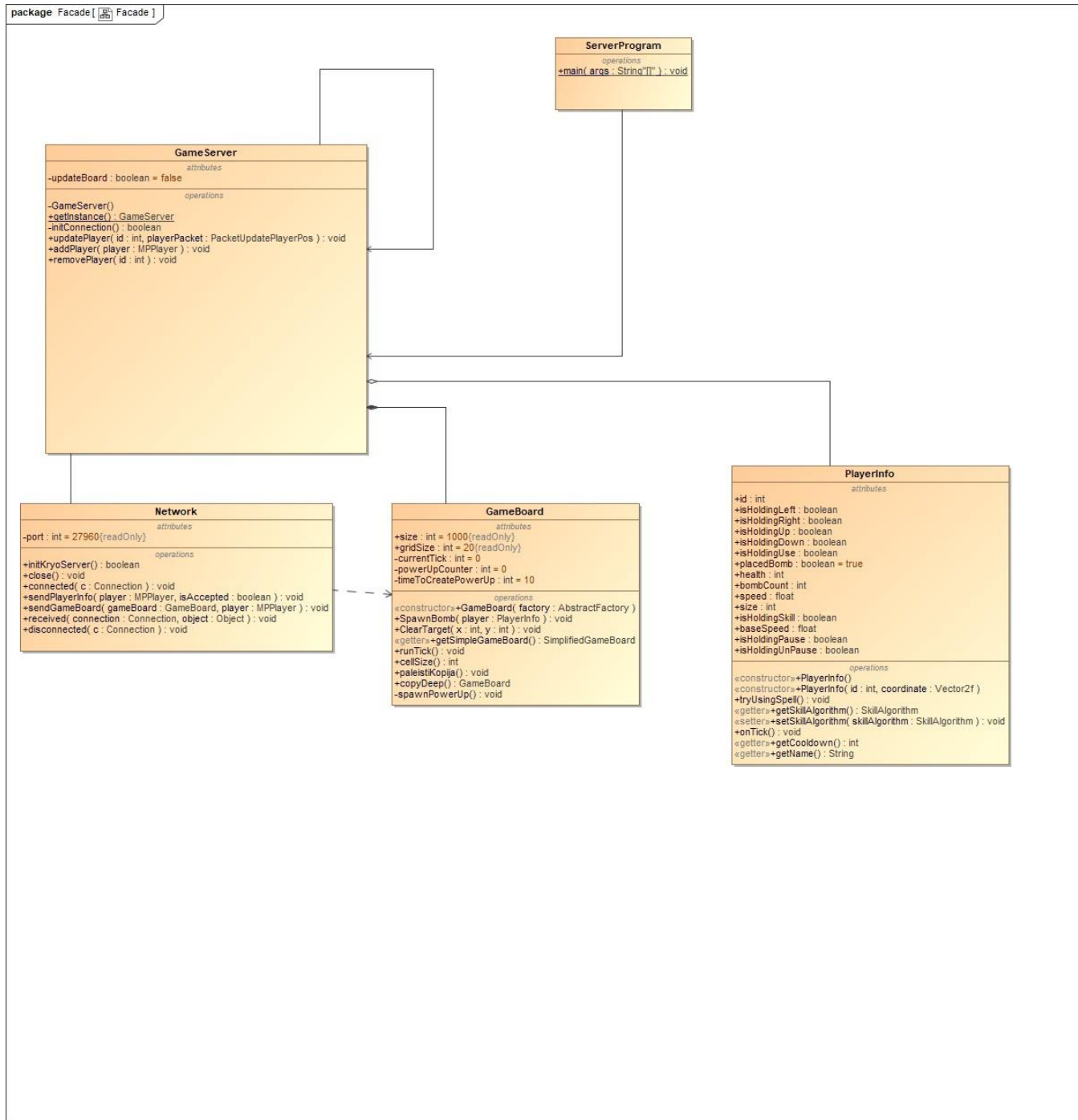
```

```

172         for (int y = 0; y < this.gridSize; y++)
173         {
174             if (this.objects[x][y].isDead) {
175
176
177
178                 if (this.objects[x][y] instanceof PowerUp)
179                 {
180                     System.out.println("Sunaikinau powerUpa");
181                     powerUpCounter -= 1;
182                 }
183
184                 this.ClearTarget(x, y);
185
186
187
188             }
189
190             objects[x][y].onTick();
191         }
192     }
193 }
194
195 public int cellSize()
196 {
197     return size / gridSize;
198 }
199 }

```

5.1.6. Facade



5.10 pav. Facade diagrama

Naudojant Facade mūsų serverio žaidimo logikos funkcionalumas yra prieinamas prie GameServer klasės kuri yra atitraukta nuo viso kito programos funkcionalumo, taigi jeigu reikėtų implemetuoti papildomas sub sistemas, kaip monitoring arba serverio GUI, tai būtų lengviau.


```
1 package server;
2
3 public class ServerProgram
4 {
5     public static void main(String [] args)
6     {
7         GameServer.getInstance();
8     }
9 }
```

```
1 package server;
2
3 import java.util.HashMap;
4 import java.util.Map;
5
6 import shared.Vector2f;
7 import shared.PacketUpdatePlayerPos;
8
9 class GameServer
10 {
11     private static GameServer gameServer = null;
12
13     protected volatile GameBoard gameBoard;
14     protected volatile Map<Integer, MPPlayer> players;
15     protected volatile Network network;
16     private GameCycleThread thread;
17     private Stage1Factory stage1factory;
18     private boolean updateBoard = false;
19 }
```

```

20 private GameServer ()
21 {
22     //Init Connection
23     if (!initConnection())
24     {
25         System.err.println("ERROR Connecting to host");
26         return;
27     }
28
29     this.players = new HashMap<Integer , MPPlayer>();
30     this.stage1factory = new Stage1Factory();
31     this.gameBoard = new GameBoard(stage1factory);
32
33     this.thread = new GameCycleThread();
34     this.thread.start();
35 }
36
37 public static GameServer getInstance()
38 {
39     if (gameServer == null)
40     {
41         gameServer = new GameServer();
42     }
43     return gameServer;
44 }
45
46 private boolean initConnection()
47 {
48     this.network = new Network();

```

```

49
50     if (!this.network.initKryoServer())
51     {
52         return false;
53     }
54
55     return true;
56 }
57
58 public void updatePlayer(int id, PacketUpdatePlayerPos playerPacket)
59 {
60     MPPlayer player = players.get(id);
61     if (player != null)
62     {
63         player.isHoldingUp = playerPacket.isHoldingUp != null ?
64         playerPacket.isHoldingUp : player.isHoldingUp;
65         player.isHoldingDown = playerPacket.isHoldingDown != null ?
66         playerPacket.isHoldingDown : player.isHoldingDown;
67         player.isHoldingLeft = playerPacket.isHoldingLeft != null ?
68         playerPacket.isHoldingLeft : player.isHoldingLeft;
69         player.isHoldingRight = playerPacket.isHoldingRight != null ?
70         playerPacket.isHoldingRight : player.isHoldingRight;
71         player.isHoldingUse = playerPacket.isHoldingUse != null ?
72         playerPacket.isHoldingUse : player.isHoldingUse;
73         player.isHoldingSkill = playerPacket.isHoldingSkill != null ?
74         playerPacket.isHoldingSkill : player.isHoldingSkill;
75         players.put(player.id, player);
76     }
77 }

```

```

72
73 public void addPlayer(MPPlayer player)
74 {
75     this.players.put(player.c.getID(), player);
76     this.network.sendGameBoard(gameBoard, player);
77
78 }
79
80 public void removePlayer(int id)
81 {
82
83     players.remove(id);
84 }
85
86
87 private class GameCycleThread extends Thread
88 {
89     volatile boolean isGameRunning = true;
90     private final int gameSpeed = 16; //The lower the number the
91     faster the game is
92
93     public GameCycleThread()
94     {
95         this.isGameRunning = true;
96     }
97
98     public void run()
99     {
100         while (this.isGameRunning)

```

```

100     {
101         try
102         {
103             //Probably should use Timer instead
104             Thread.sleep(gameSpeed);
105             this.update();
106         }
107         catch (InterruptedException e)
108         {
109             e.printStackTrace();
110             this.stopGame();
111         }
112     }
113 }
114
115 public void stopGame()
116 {
117     this.isGameRunning = false;
118
119     //network should probably be closed by the parent
120     network.close();
121 }
122
123 private void update()
124 {
125     updatePlayers();
126     gameBoard.runTick();
127 }
128

```

```

129     private void updatePlayers ()
130     {
131         for (MPPlayer p : players.values())
132         {
133             if (p.isHoldingPause )
134             {
135
136             }
137
138             if (p.isHoldingSkill)
139             {
140                 p.tryUsingSpell();
141             }
142
143             p.onTick();
144
145             if (p.isHoldingUse )
146             {
147                 gameBoard.SpawnBomb(p);
148             }
149
150             p.coordinate = checkCollision(p);
151
152             network.sendGameBoard(gameBoard, p);
153             network.sendPlayerInfo(p, true);
154         }
155     }
156
157     private Vector2f checkCollision(MPPlayer p)

```

```

158     {
159
160         Vector2f coordsAfterMove = new Vector2f(p.coordinate.x, p.
coordinate.y);
161
162         float padding = 0.001f;
163         float cellSize = gameBoard.cellSize();
164
165         boolean moveX = true;
166         boolean moveY = true;
167
168         if (p.isHoldingLeft)
169         {
170             coordsAfterMove.x -= p.speed;
171         }
172
173         if (p.isHoldingRight)
174         {
175
176             coordsAfterMove.x += p.speed;
177         }
178
179         boolean collidingLeft = ((int)coordsAfterMove.x / cellSize -
padding) < ((int)p.coordinate.x / cellSize);
180         boolean collidingRight = (((int)coordsAfterMove.x + p.size +
padding) / cellSize) > (((int)p.coordinate.x + p.size) / cellSize);
181         boolean isCollidingX = collidingLeft || collidingRight;
182         moveX = !(coordsAfterMove.x <= 0 || coordsAfterMove.x >=
gameBoard.size - p.size);

```

```

183 //Some smoothing when going around edges would be nice
184 if (isCollidingX && moveX)
185 {
186     int x = 0, y = 0, y1 = 0;
187     if (collidingLeft)
188     {
189         x = (int) ((coordsAfterMove.x) / cellSize);
190         y = (int) (p.coordinate.y / cellSize);
191         y1 = (int) ((p.coordinate.y + p.size) / cellSize);
192
193     }
194     if (collidingRight)
195     {
196         x = (int) ((coordsAfterMove.x + p.size) / cellSize);
197         y = (int) (p.coordinate.y / cellSize);
198         y1 = (int) ((p.coordinate.y + p.size) / cellSize);
199
200     }
201
202     if (y == y1)
203     {
204         moveX = gameBoard.objects[x][y].isWalkable;
205         gameBoard.objects[x][y].onStep(p);
206     }
207     else
208     {
209         moveX = gameBoard.objects[x][y].isWalkable && gameBoard.
objects[x][y1].isWalkable;
210         gameBoard.objects[x][y].onStep(p);

```



```

211         gameBoard . objects [ x ] [ y1 ] . onStep ( p ) ;
212     }
213 }
214
215
216     if ( p . isHoldingUp )
217     {
218         coordsAfterMove . y += p . speed ;
219     }
220
221     if ( p . isHoldingDown )
222     {
223         coordsAfterMove . y -= p . speed ;
224     }
225
226     boolean collidingUp = (((int)coordsAfterMove . y + p . size - padding
) / cellSize) > (((int)p . coordinate . y + p . size) / cellSize);
227     boolean collidingDown = ((int)coordsAfterMove . y / cellSize +
padding) < ((int)p . coordinate . y / cellSize);
228     boolean isCollidingY = collidingUp || collidingDown ;
229     moveY = !( coordsAfterMove . y <= 0 || coordsAfterMove . y >=
gameBoard . size - p . size );
230
231     //Some smoothing when going around edges would be nice
232     if ( isCollidingY && moveY )
233     {
234         int x = 0, x1 = 0, y = 0;
235
236         if ( collidingUp )

```

```

237     {
238         y = (int) ((coordsAfterMove.y + p.size) / cellSize);
239         x = (int) (p.coordinate.x / cellSize);
240         x1 = (int) ((p.coordinate.x + p.size) / cellSize);
241
242     }
243     if (collidingDown)
244     {
245         y = (int) ((coordsAfterMove.y) / cellSize);
246         x = (int) (p.coordinate.x / cellSize);
247         x1 = (int) ((p.coordinate.x + p.size) / cellSize);
248     }
249
250     if (x == x1)
251     {
252         moveY = gameBoard.objects[x][y].isWalkable;
253         gameBoard.objects[x][y].onStep(p);
254     }
255     else
256     {
257         moveY = gameBoard.objects[x][y].isWalkable && gameBoard.
objects[x1][y].isWalkable;
258         gameBoard.objects[x][y].onStep(p);
259         gameBoard.objects[x1][y].onStep(p);
260     }
261 }
262
263 coordsAfterMove.x = moveX ? coordsAfterMove.x : p.coordinate.x;
264 coordsAfterMove.y = moveY ? coordsAfterMove.y : p.coordinate.y;

```

```

265
266     return coords After Move ;
267
268     }
269
270 }
271
272 }

```

```

1 package server ;
2
3 import org.lwjgl.system.CallbackI.P;
4
5 import com.esotericsoftware.kryonet.Connection ;
6 import com.esotericsoftware.kryonet.Listener ;
7 import com.esotericsoftware.kryonet.Server ;
8
9 import shared.PacketUpdatePlayerPos ;
10 import shared.SimplifiedGameBoard ;
11 import shared.SimplifiedGameObject ;
12 import shared.ObjectType ;
13 import shared.PacketAddPlayer ;
14 import shared.PacketRemovePlayer ;
15 import shared.PacketUpdateGameBoard ;
16 import shared.Vector2f ;
17
18 public class Network extends Listener
19 {
20     private Server server ;

```

```

21 private final int port = 27960;
22
23 public boolean initKryoServer()
24 {
25     try
26     {
27         this.server = new Server(131072, 16384);
28         this.server.getKryo().register(PacketUpdatePlayerPos.class);
29         this.server.getKryo().register(PacketAddPlayer.class);
30         this.server.getKryo().register(PacketRemovePlayer.class);
31         this.server.getKryo().register(PacketUpdateGameBoard.class);
32         this.server.getKryo().register(SimplifiedGameBoard.class);
33         this.server.getKryo().register(ObjectType.class);
34         this.server.getKryo().register(SimplifiedGameObject.class);
35         this.server.getKryo().register(SimplifiedGameObject[].class);
36         this.server.getKryo().register(SimplifiedGameObject[][].class);
37         this.server.getKryo().register(Vector2f.class);
38         this.server.bind(this.port, this.port);
39         this.server.addListener(this);
40         this.server.start();
41         System.out.println("The server is ready");
42         return true;
43     }
44     catch (Exception e)
45     {
46         e.printStackTrace();
47         return false;
48     }
49 }

```

```

50
51 public void close()
52 {
53     this.server.close();
54 }
55
56 public void connected(Connection c)
57 {
58     MPPlayer player = new MPPlayer();
59     player.c = c;
60
61     PacketAddPlayer packet = new PacketAddPlayer();
62     packet.id = c.getID();
63     this.server.sendToAllExceptTCP(c.getID(), packet);
64     GameServer.getInstance().addPlayer(player);
65     System.out.println("Connection received.");
66 }
67
68 public void sendPlayerInfo(MPPlayer player, boolean isAccepted)
69 {
70     PacketUpdatePlayerPos packet = new PacketUpdatePlayerPos(player.c.
71     getID(), player);
72     packet.accepted = isAccepted;
73     this.server.sendToAllUDP(packet);
74 }
75
76 public void sendGameBoard(GameBoard gameBoard, MPPlayer player)
77 {

```

```

78     Simplified Game Board  simpleGameboard = gameBoard .getSimpleGameBoard
    ();
79     if (player == null)
80     {
81         PacketUpdateGameBoard packet = new PacketUpdateGameBoard (
    simpleGameboard );
82         this . server . sendToAllUDP ( packet );
83     }
84     else
85     {
86         PacketUpdateGameBoard packet = new PacketUpdateGameBoard (
    simpleGameboard );
87         this . server . sendToUDP ( player . c . getID () , packet );
88     }
89 }
90
91 public void received ( Connection connection , Object object )
92 {
93     if ( object instanceof PacketUpdatePlayerPos )
94     {
95         PacketUpdatePlayerPos packet = ( PacketUpdatePlayerPos ) object ;
96         GameServer . getInstance () . updatePlayer ( connection . getID () , packet )
    ;
97         System . out . println ( "Received coordinate packet " );
98     }
99 }
100
101 public void disconnected ( Connection c )
102 {

```

```

103 PacketRemovePlayer packet = new PacketRemovePlayer ();
104 packet.id = c.getID ();
105 this.server.sendToAllExceptTCP (c.getID (), packet );
106 GameServer.getInstance ().removePlayer (c.getID ());
107 System.out.println("Connection dropped.");
108 }
109 }

```

```

1 package server;
2
3 import java.util.ArrayList;
4 import java.util.Random;
5
6 import shared.*;
7
8 public class GameBoard implements Cloneable {
9
10     public final int size = 1000;
11     public final int gridSize = 20;
12     public GameObject[][] objects;
13     private BombObserver bombObserver;
14     private AbstractFactory factory;
15
16     private int currentTick = 0;
17     private int powerUpCounter = 0;
18     private int timeToCreatePowerUp = 10;
19
20     private IStageBuilder stage1builder;
21     private IStageBuilder stage2builder;

```

```

22     private IStageBuilder stage3builder;
23
24     public GameBoard( AbstractFactory factory )
25     {
26         this.factory = factory;
27         this.objects = new GameObject[ gridSize ][ gridSize ];
28         this.stage1builder = new Stage1Builder( gridSize );
29         this.stage2builder = new Stage2Builder( gridSize );
30         this.stage3builder = new Stage3Builder( gridSize );
31
32         StageDirector stageDirector = new StageDirector( stage3builder );
33
34         stageDirector.makeStage();
35
36         Stage stage1 = stageDirector.getStage();
37
38         for( Coordinates kor : stage1.getGrounds() )
39             this.objects[ kor.getX() ][ kor.getY() ] = this.factory.
createGround();
40
41         int kiekis = 76;
42         boolean sunaikinama = false;
43
44         for( Coordinates kor : stage1.getWalls() ){
45
46             this.objects[ kor.getX() ][ kor.getY() ] = this.factory.
createWall( sunaikinama );
47             kiekis -=1;
48             if( kiekis == 0 ) sunaikinama = true;

```



```
49     }
50
51
52
53     bombObserver = new BombObserver ( this );
54
55
56     this . factory . SetBombObserver ( bombObserver );
57
58
59
60
61
62     //test
63
64     //TODO wrong but ok for now
65     this . objects [ 17 ][ 3 ] = this . factory . createTrap () ;
66
67     this . objects [ 15 ][ 3 ] = this . factory . createTrap () ;
68     Trap modified = (Trap) this . objects [ 15 ][ 3 ];
69     modified . setTrapeffect ( new DamageTrap ( new ConcreteTrap () ) );
70     this . objects [ 15 ][ 3 ] = modified ;
71
72     this . objects [ 13 ][ 3 ] = this . factory . createTrap () ;
73     Trap modified1 = (Trap) this . objects [ 13 ][ 3 ];
74     modified1 . setTrapeffect ( new TeleportTrap ( new ConcreteTrap () ) );
75     this . objects [ 13 ][ 3 ] = modified1 ;
76
77     this . objects [ 11 ][ 3 ] = this . factory . createTrap () ;
```

```

87      Trap modified2 = (Trap) this.objects[11][3];
88      modified2.setTrapeffect(new SlowTrap (new ConcreteTrap()));
89      this.objects[11][3] = modified2;
90
91      this.objects[9][3] = this.factory.createTrap();
92      Trap modified3 = (Trap) this.objects[9][3];
93      modified3.setTrapeffect(new DamageTrap (new TeleportTrap (new
SlowTrap (new ConcreteTrap()))));
94      this.objects[9][3] = modified3;
95
96  }
97
98  public void paleistiKopija(){
99      GameBoard copy = copyDeep();
100
101  }
102
103  public GameBoard copyDeep(){
104
105      try {
106          return (GameBoard) this.clone();
107      } catch (CloneNotSupportedException e) {
108          e.printStackTrace();
109          return null;
110      }
111  }

```

```

106
107     public void SpawnBomb( PlayerInfo  player)
108     {
109         int x = Math.round( player.coordinate.x/( size/gridSize));
110         int y = Math.round( player.coordinate.y/( size/gridSize));
111         this.objects[x][y] = this.factory.createBomb( player.id);
112
113     }
114
115     public void ClearTarget( int x, int y){
116         this.objects[x][y] = this.factory.createGround();
117         System.out.println("Removing bomb from location " + x + " " + y
118     );
119     }
120
121     public SimplifiedGameBoard  getSimpleGameBoard ()
122     {
123         SimplifiedGameBoard  simpleGameboard = new SimplifiedGameBoard( this.
124         size , this.gridSize);
125
126         for (int i = 0; i < this.gridSize; i ++)
127         {
128             for (int j = 0; j < this.gridSize; j ++)
129             {
130                 simpleGameboard.objects[i][j] = new SimplifiedGameObject
131                 ( this.objects[i][j].color , ObjectType.GROUND);
132             }
133         }
134
135         return simpleGameboard;

```

```
132     }
133
134     private void spawnPowerUp(){
135         Random rand = new Random();
136
137         if(powerUpCounter > 4)
138             return;
139
140         while (true){
141             int x = rand.nextInt((19 - 0) + 1) + 0;
142             int y = rand.nextInt((19 - 0) + 1) + 0;
143
144
145             if(this.objects[x][y] instanceof Ground){
146                 this.objects[x][y] = this.factory.createPowerUp();
147                 powerUpCounter+=1;
148                 break;
149             }
150
151         }
152
153
154
155
156
157
158
159
160     }
```

```

161
162
163     public void runTick ()
164     {
165         currentTick +=1;
166
167         if (currentTick % (60 * timeToCreatePowerUp) == 0) // kas 10s
168             sukurti nauja powerUp
169             spawnPowerUp () ;
170
171         for (int x = 0; x < this.gridSize; x++)
172         {
173             for (int y = 0; y < this.gridSize; y++)
174             {
175                 if (this.objects[x][y].isDead) {
176
177
178                     if (this.objects[x][y] instanceof PowerUp)
179                     {
180                         System.out.println("Sunaikinau powerUpa");
181                         powerUpCounter -=1;
182                     }
183
184                     this.ClearTarget(x, y);
185
186
187
188                 }

```

```

189
190         objects[x][y].onTick();
191     }
192 }
193 }
194
195 public int cellSize()
196 {
197     return size / gridSize;
198 }
199 }

```

```

1 package server;
2
3 import shared.Vector2f;
4
5 public class PlayerInfo
6 {
7     public int id;
8     public Vector2f coordinate;
9     public boolean isHoldingLeft;
10    public boolean isHoldingRight;
11    public boolean isHoldingUp;
12    public boolean isHoldingDown;
13    public boolean isHoldingUse;
14    public boolean isHoldingSkill;
15    public boolean isHoldingPause;
16    public boolean isHoldingUnPause;
17    public boolean placedBomb = true;

```

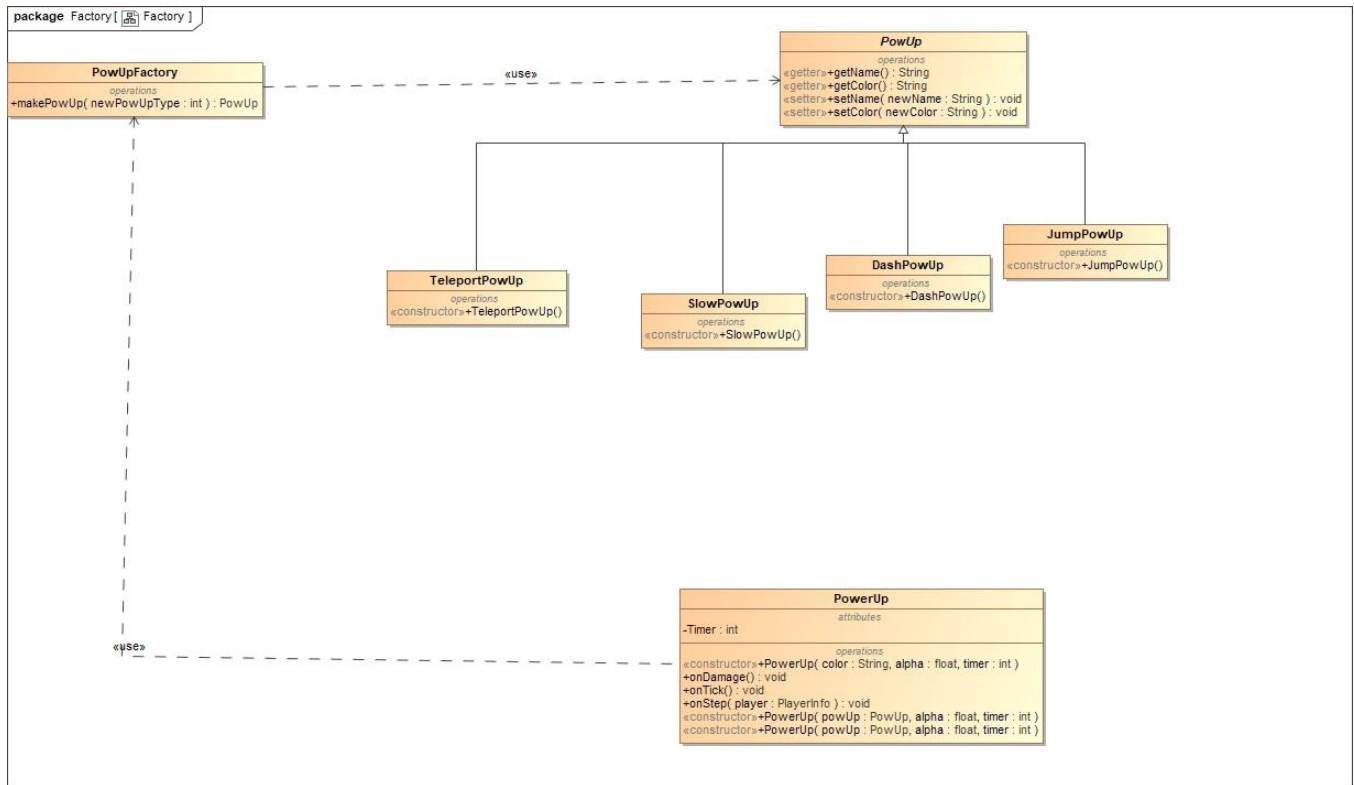
```
18 public int health ;
19 public float speed ;
20 public float baseSpeed ;
21 public int size ;
22 public int bombCount ;
23
24 private SkillAlgorithm skillAlgorithm ;
25
26 public PlayerInfo ()
27 {
28     this.coordinate = new Vector2f () ;
29     this.coordinate.x = 400 ;
30     this.coordinate.y = 400 ;
31     this.isHoldingLeft = false ;
32     this.isHoldingRight = false ;
33     this.isHoldingUp = false ;
34     this.isHoldingDown = false ;
35     this.isHoldingPause = false ;
36     this.isHoldingUnPause = false ;
37     this.skillAlgorithm = new DashSkill () ;
38     this.size = 40 ;
39     this.speed = 2.5f ;
40     this.baseSpeed = 2.5f ;
41     this.health = 3 ;
42     this.bombCount = 2 ;
43     //this.playerStats = new ConcretePlayer () ;
44 }
45
46 public PlayerInfo (int id , Vector2f coordinate )
```

```
47 {
48     this.id = id ;
49     this.coordinate = coordinate ;
50     this.isHoldingLeft = false ;
51     this.isHolding Right = false ;
52     this.isHoldingUp = false ;
53     this.isHoldingDown = false ;
54     this.isHolding Use = false ;
55     this.isHoldingPause = false ;
56     this.isHoldingUnPause = false ;
57     this.size = 40;
58     this.speed = 2.5 f;
59     this.base Speed = 2.5 f;
60     this.health = 3;
61     this.bombCount = 2;
62     //Test
63 }
64
65 public SkillAlgorithm getSkillAlgorithm ()
66 {
67     return skillAlgorithm ;
68 }
69
70 public void setSkillAlgorithm (SkillAlgorithm skillAlgorithm)
71 {
72     this.skillAlgorithm = skillAlgorithm ;
73 }
74
75
```



```
76 public void onTick ()
77 {
78     this.skillAlgorithm.onTick (this);
79 }
80
81 public void tryUsingSpell ()
82 {
83     this.skillAlgorithm.useSkill (this);
84 }
85
86 public int getCooldown ()
87 {
88     return this.skillAlgorithm.getCooldown ();
89 }
90
91 public String getName ()
92 {
93     return this.skillAlgorithm.getName ();
94 }
95 }
```

5.1.7. Factory



5.11 pav. Factory diagrama

Factory šablono poreikis atsirado norint kad dinamiškai keistusį žaidėjo įgudžiai.

```
1 package server;
2
3 public class PowUpFactory {
4     public PowUp makePowUp(int newPowUpType) {
5         PowUp newPowUp = null;
6
7         if(1 == newPowUpType)
8             return new JumpPowUp();
9
10        else if(2 == newPowUpType)
11            return new DashPowUp();
```

```
12
13     else if (3 == newPowUpType)
14         return new SlowPowUp();
15
16     else if (4 == newPowUpType)
17         return new TeleportPowUp();
18
19     return newPowUp;
20 }
21
22
23 }
```

```
1 package server;
2
3 public abstract class PowUp {
4
5     private String name;
6     private String color;
7
8
9
10    public String getName() {
11        return name;
12    }
13
14    public String getColor() {
15        return color;
16    }
17 }
```

```
17
18     public void setName(String newName){
19         name = newName;
20     }
21
22     public void setColor(String newColor){
23         color = newColor;
24     }
25
26
27 }
```

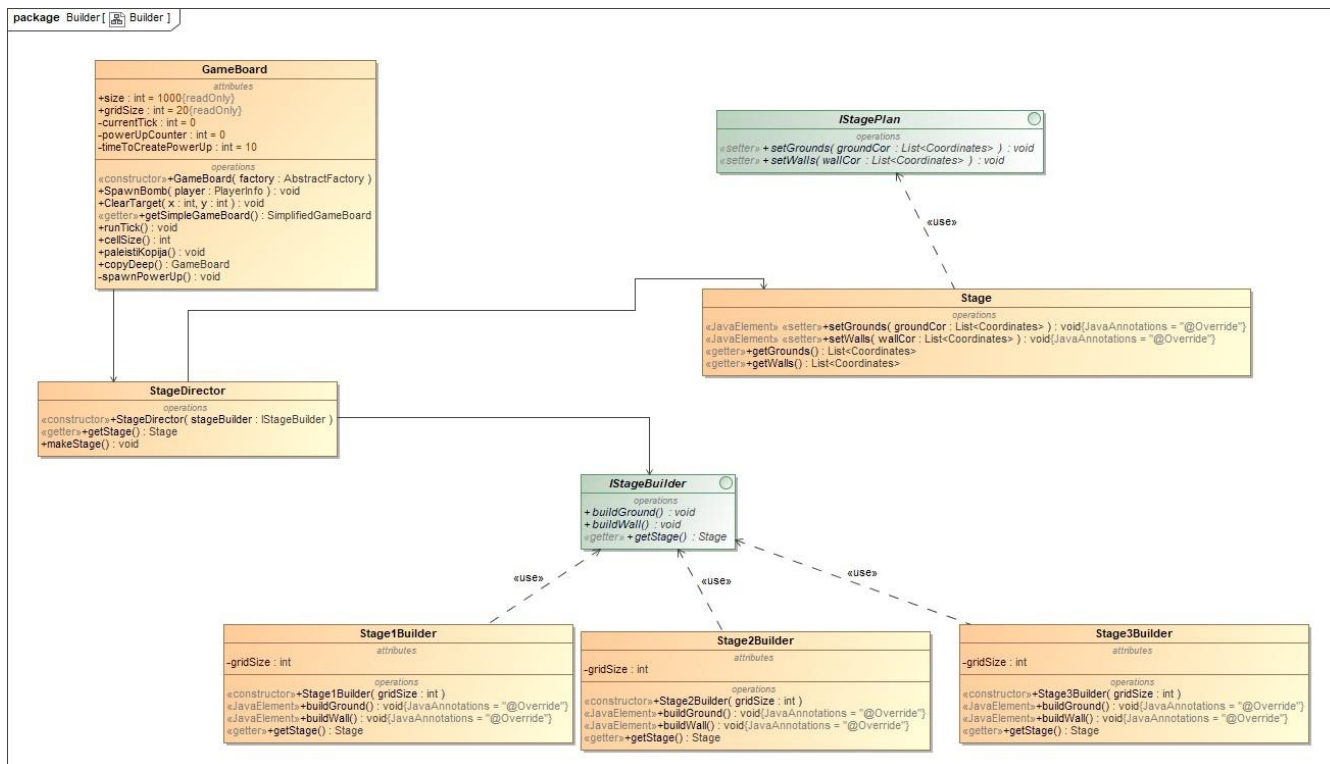
```
1 package server;
2
3 public class SlowPowUp extends PowUp{
4     public SlowPowUp(){
5         setName("Slow");
6         setColor("#0728ab");
7     }
8 }
```

```
1 package server;
2
3 public class DashPowUp extends PowUp{
4
5     public DashPowUp(){
6         setName("Dash");
7         setColor("#ccb23");
8     }
9 }
```

```
1 package server;
2
3 public class JumpPowUp extends PowUp{
4     public JumpPowUp(){
5         setName ( "Jump" );
6         setColor ( "#ab4907" );
7     }
8 }
```

```
1 package server;
2
3 public class TeleportPowUp extends PowUp{
4
5     public TeleportPowUp () {
6         setName ( "Teleport " );
7         setColor ( "#7b219e" );
8     }
9 }
```

5.1.8. Builder



5.12 pav. Builder diagrama

Builder šablono pereikis atsirado norint palengvinti kiekvieno žaidimo lygio kurimą.

```
1 package server ;
2
3 import java . util . List ;
4
5 public class Stage implements ISStagePlan {
6
7
8     private List <Coordinates> grounCor ;
9     private List <Coordinates> wallCor ;
10
11     @Override
```

```
12     public void setGrounds ( List<Coordinates> groundCor ) {
13         this.grounCor = groundCor ;
14     }
15
16     @Override
17     public void setWalls ( List<Coordinates> wallCor ) {
18         this.wallCor = wallCor ;
19     }
20
21     public List<Coordinates> getGrounds () {
22         return this.grounCor ;
23     }
24
25     public List<Coordinates> getWalls () {
26         return this.wallCor ;
27     }
28
29
30
31
32
33
34
35
36 }
```

```
1 package server ;
2
3 public class StageDirector {
```

```

4     private IStageBuilder stageBuilder;
5
6     public StageDirector(IStageBuilder stageBuilder){
7         this.stageBuilder = stageBuilder;
8     }
9
10    public Stage getStage(){
11        return this.stageBuilder.getStage();
12    }
13
14
15    public void makeStage(){
16        this.stageBuilder.buildWall();
17        this.stageBuilder.buildGround();
18    }
19 }

```

```

1 package server;
2
3 import java.util.ArrayList;
4 import java.util.List;
5
6 public class Stage1Builder implements IStageBuilder{
7     private Stage stage;
8     private int gridSize;
9
10    public Stage1Builder(int gridSize){
11        this.stage = new Stage();
12        this.gridSize = gridSize;

```



```

13     }
14
15     @Override
16     public void buildGround () {
17         List<Coordinates> ground = new ArrayList<Coordinates>();
18
19         for(int x = 0; x<gridSize; x++){
20             for(int y = 0; y<gridSize; y++){
21                 if(x != 0 && x != gridSize-1 && y != 0 && y != gridSize
22 -1)
23                     ground.add(new Coordinates (x, y));
24             }
25         }
26
27         stage.setGrounds (ground);
28     }
29
30     @Override
31     public void buildWall () {
32         List<Coordinates> walls = new ArrayList<Coordinates>();
33
34         for(int x = 0; x<gridSize; x++){
35             for(int y = 0; y<gridSize; y++){
36                 if(x == 0 || x == gridSize-1 || y == 0 || y == gridSize
37 -1)
38                     walls.add(new Coordinates (x, y));
39             }
40         }

```

```

40
41     System.out.println("Koks dydis Pries: " + walls.size());
42
43     walls.add(new Coordinates(5, 4));
44     walls.add(new Coordinates(5, 6));
45     walls.add(new Coordinates(6, 5));
46
47     walls.add(new Coordinates(15, 16));
48     walls.add(new Coordinates(15, 14));
49     walls.add(new Coordinates(14, 15));
50
51     for(int x = 1; x < 19; x++){
52         walls.add(new Coordinates(x, 11));
53     }
54
55     System.out.println("Koks dydis Po: " + walls.size());
56
57     stage.setWalls(walls);
58 }
59
60 public Stage getStage(){
61     return this.stage;
62 }
63
64 }

```

```

1 package server;
2
3 import java.util.ArrayList;

```

```

4 import java.util.List;
5
6 public class Stage2Builder implements IStageBuilder{
7     private Stage stage;
8     private int gridSize;
9
10    public Stage2Builder(int gridSize){
11        this.stage = new Stage();
12        this.gridSize = gridSize;
13    }
14
15    @Override
16    public void buildGround() {
17        List<Coordinates> ground = new ArrayList<Coordinates>();
18
19        for(int x = 0; x<gridSize; x++){
20            for(int y = 0; y<gridSize; y++){
21                if(x != 0 && x != gridSize-1 && y != 0 && y != gridSize
22                -1)
23                    ground.add(new Coordinates(x, y));
24            }
25        }
26
27        stage.setGrounds(ground);
28    }
29
30    @Override
31    public void buildWall() {

```

```

32 List<Coordinates> walls = new ArrayList<Coordinates>();
33
34 for(int x = 0; x<gridSize; x++){
35     for(int y = 0; y<gridSize; y++){
36         if(x == 0 || x == gridSize -1 || y == 0 || y == gridSize
-1)
37             walls.add(new Coordinates(x, y));
38     }
39 }
40
41 System.out.println("Koks dydis Pries: " + walls.size());
42
43
44
45
46 for(int i = 7; i<11; i++){
47     walls.add(new Coordinates(i, 9));
48     walls.add(new Coordinates(i, 13));
49     walls.add(new Coordinates(i, 17));
50 }
51
52 for(int i =10; i<17; i++){
53     if(i != 13){
54         walls.add(new Coordinates(6, i));
55         walls.add(new Coordinates(11, i));
56     }
57 }
58
59

```

```

60
61
62
63
64     System.out.println("Koks dydis Po: " + walls.size());
65
66     stage.setWalls(walls);
67 }
68
69 public Stage getStage(){
70     return this.stage;
71 }
72
73 }

```

```

1 package server;
2
3 import java.util.ArrayList;
4 import java.util.List;
5
6 public class Stage3Builder implements IStageBuilder{
7     private Stage stage;
8     private int gridSize;
9
10    public Stage3Builder(int gridSize){
11        this.stage = new Stage();
12        this.gridSize = gridSize;
13    }
14

```

```
@Override
```

```
public void buildGround () {
```

```
    List<Coordinates> ground = new ArrayList<Coordinates>();
```

```
    for(int x = 0; x<gridSize; x++){
```

```
        for(int y = 0; y<gridSize; y++){
```

```
            if(x != 0 && x != gridSize-1 && y != 0 && y != gridSize-1)
```

```
                ground.add(new Coordinates(x, y));
```

```
        }
```

```
    }
```

```
    stage.setGrounds(ground);
```

```
}
```

```
@Override
```

```
public void buildWall () {
```

```
    List<Coordinates> walls = new ArrayList<Coordinates>();
```

```
    for(int x = 0; x<gridSize; x++){
```

```
        for(int y = 0; y<gridSize; y++){
```

```
            if(x == 0 || x == gridSize-1 || y == 0 || y == gridSize-1)
```

```
                walls.add(new Coordinates(x, y));
```

```
        }
```

```
    }
```

```
    System.out.println("Koks dydis Pries: " + walls.size());
```

42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70

```
walls.add(new Coordinates (3, 14 -1));  
walls.add(new Coordinates (4, 14 -1));  
walls.add(new Coordinates (5, 14 -1));  
walls.add(new Coordinates (6, 14 -1));  
walls.add(new Coordinates (3, 18 -1));  
walls.add(new Coordinates (4, 18 -1));  
walls.add(new Coordinates (5, 18 -1));  
walls.add(new Coordinates (6, 18 -1));  
walls.add(new Coordinates (3, 17 -1));  
walls.add(new Coordinates (3, 16 -1));  
walls.add(new Coordinates (3, 15 -1));  
walls.add(new Coordinates (5, 16 -1));  
walls.add(new Coordinates (6, 16 -1));  
walls.add(new Coordinates (6, 15 -1));  
  
walls.add(new Coordinates (8, 14 -1));  
walls.add(new Coordinates (8, 15 -1));  
walls.add(new Coordinates (8, 16 -1));  
walls.add(new Coordinates (8, 17 -1));  
walls.add(new Coordinates (11, 14 -1));  
walls.add(new Coordinates (11, 15 -1));  
walls.add(new Coordinates (11, 16 -1));  
walls.add(new Coordinates (11, 17 -1));
```

```
71 walls.add(new Coordinates (9, 16-1));
72 walls.add(new Coordinates (10, 16-1));
73 walls.add(new Coordinates (9, 18-1));
74 walls.add(new Coordinates (10, 18-1));
75
76
77
78 walls.add(new Coordinates (13, 14-1));
79 walls.add(new Coordinates (14, 14-1));
80 walls.add(new Coordinates (15, 14-1));
81 walls.add(new Coordinates (16, 14-1));
82
83 walls.add(new Coordinates (13, 15-1));
84 walls.add(new Coordinates (13, 16-1));
85 walls.add(new Coordinates (13, 17-1));
86 walls.add(new Coordinates (13, 18-1));
87
88
89 for(int i = 8; i<12; i++){
90     walls.add(new Coordinates (i, 11-1));
91     walls.add(new Coordinates (i, 8-1));
92     walls.add(new Coordinates (i, 5-1));
93 }
94
95 walls.add(new Coordinates (8, 10-1));
96 walls.add(new Coordinates (8, 9-1));
97 walls.add(new Coordinates (11, 10-1));
98 walls.add(new Coordinates (11, 9-1));
99 walls.add(new Coordinates (11, 7-1));
```



```
100     walls.add(new Coordinates (11, 6-1));
101
102
103     System.out.println("Koks dydis Po: " + walls.size());
104
105     stage.setWalls (walls);
106 }
107
108 public Stage getStage () {
109     return this.stage;
110 }
111
112
113 }
```

```
1 package server;
2
3 public interface IStageBuilder {
4     public void buildGround ();
5     public void buildWall ();
6
7     public Stage getStage ();
8 }
```

```
1 package server;
2
3 import java.util.List;
4
5 public interface IStagePlan {
6
```

```

7      public void setGrounds ( List<Coordinates> groundCor );
8
9      public void setWalls ( List<Coordinates> wallCor );
10
11 }

```

5.1.9. Prototype



5.13 pav. Prototype diagrama

Prototype šablono poreikis atsirado norint kad butu galimi grįžtai jau ankščiau išsaugotą žaidimo stadiją, kad būtų galima į ją grįžti.

6. PROJEKTO APRAŠYMAS ANTRA DALIS

Žaidimas, kuriame strategiškai dėlioji bombas, kuriomis gali sunaikinti kliūtis bei priešus, išvenginėji spąstų bei naudoji įgytas galias kad įgautum pranašumą prieš savo varžovą.



6.1 pav. Žaidimo prototipo nuotrauka

7. ŽAIDIMO REIKALAVIMAI

ŽAIDIMO LYGIAI

Žaidimas susidės iš trijų lygių, kurie vienas nuo kito skirsis savo žaidimo strategijomis bei sudėtingumu.

PIRMASIS LYGIS

Jame bus sukurtas pasaulis, kuris susidės iš sienų, kurių kiaurai pereiti negalima, bet galima jas susprogdinti naudojant bombas, bei žemės, per kurią žaidėjas gales laisvai vaikščioti. Žaidimo tikslas nugalėti savo priešininką, taktiškai naikinant sienas. Žaidėjai turės po 3 gyvybes, viena gyvybė yra prarandama jeigu savo arba priešininko bomba sprogs šalia.

ANTRAS LYGIS

Antrame lygyje atsiranda nauja kliūtis- tai spąstai, kurie sugeneruojami sukuriant 2 lygio pasaulį. Visi spąstai atrodo taip pat tik žaidėjai nežino, ką jis gali padaryti, tik žino tą kad visi spąstai jį užsaldys. Taip pat spąstai gali žaidėja sulėtinti, nuimti gyvybę ar jį kažkur nukelti.

TREČIAS LYGIS

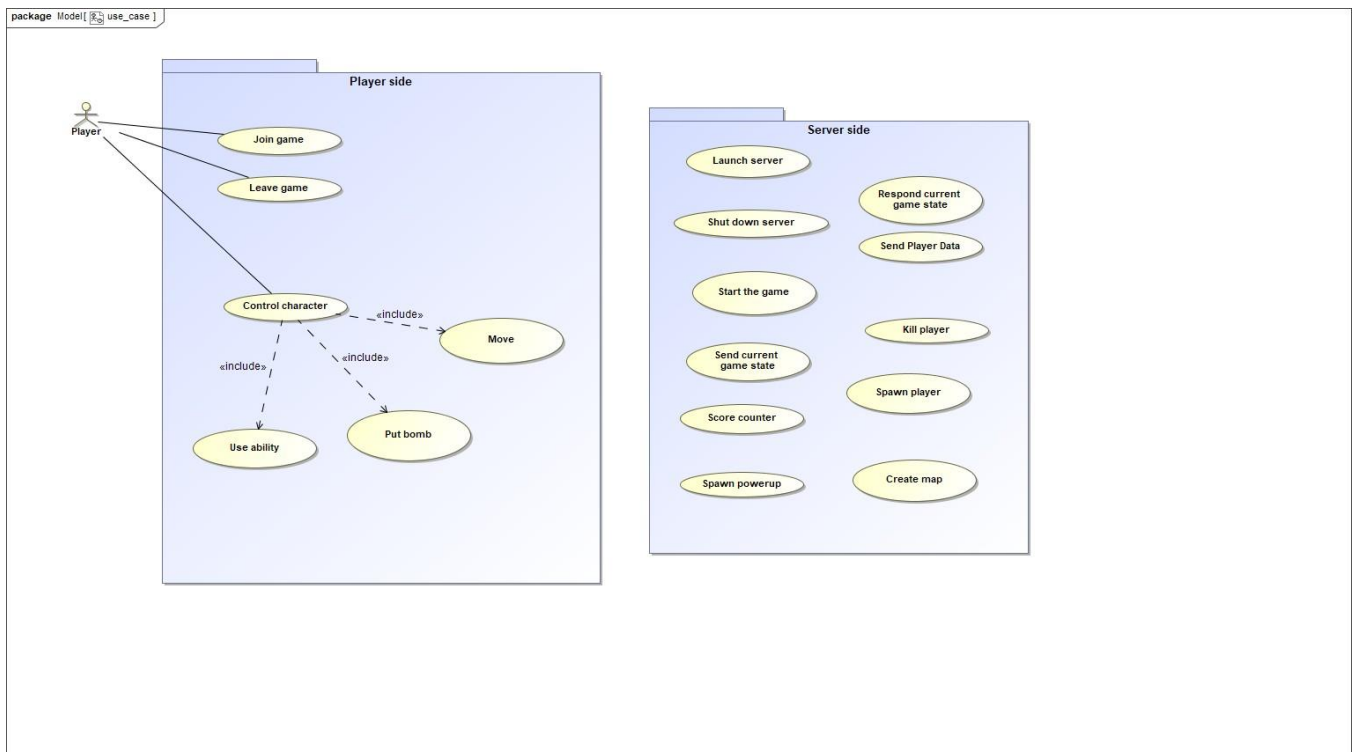
Trečiame lygyje atsiranda žaidėjo įgudis. Kai 3 lygio pasaulis yra sugeneruotas kas 10 sekundžių atsiranda žaidėjo įgudis, ant kurio užlipus žaidėjas gauna viena iš 4 pagerinimų:

- Sulėtinimas priešininko. Galima naudoti kas 30 sekundžių.
- Nusikeltas į naują vietą. Galima naudoti kas 20 sekundžių.
- Sienų peršokimas. Galima naudoti kas 15 sekundžių.
- Greitas žaidėjo paslinkimas. Galima naudoti kas 15 sekundžių.

8. PROJEKTUI NAUDOTOS TECHNOLOGIJOS

Serverio ir kliento komunikacijai naudotas Kryonet. Programos lango sukūrimui ir piešimui buvo pasitelkta lwjgl ir OpenGL 1.1 įrankiai.

9. USE CASE DIAGRAMA



4.2 pav. Use case diagrama

10. KLASIŲ DIAGRAMA



5.14 pav. Kласių diagrama

11 ŠABLONAI

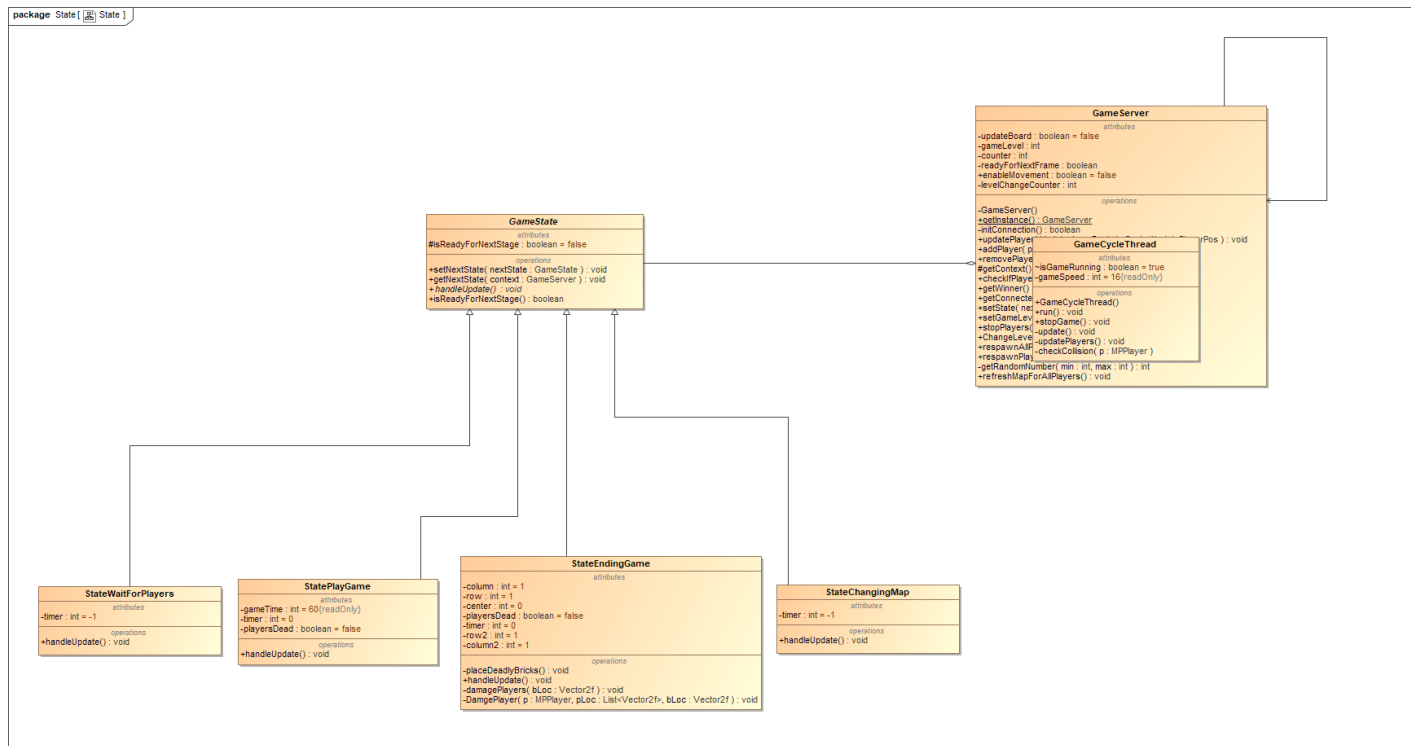
GameState

GameState šablono prirėikė norint įdėti į žaidimą skirtingas būsenas. Tai yra :

- Žaidimo laukimas
- Žaidimo vykdymas
- Žaidimo užbaigimo bandymas
- Žaidimo lentos pakeitimas į kitą

Šis šablonas puikiai padėjo įgyvendinti šią idėją.

UML diagrama:



Kodas:

```
1 package server;
2
3 public abstract class GameState
4 {
5     private GameState nextState;
6     protected boolean isReadyForNextStage = false;
7
8     public void setNextState (GameState nextState) {
9         this.nextState = nextState;
10    }
11
12    public void getNextState ( GameServer context )
13    {
14        context.setState (nextState);
15    }
16
17    public abstract void handleUpdate ();
18
19    public boolean isReadyForNextStage ()
20    {
21        return isReadyForNextStage;
22    }
23
24 }
25
```

```
1 package server;
2
```



```

3 public class StatePlayGame extends GameState
4 {
5     private final int gameTime = 60; // 1 minute
6     private int timer = 0;
7     private boolean playersDead = false;
8
9     @Override
10    public void handleUpdate ()
11    {
12        GameServer gameServer = GameServer.getInstance ();
13        this.isReadyForNextStage = false;
14        if (timer > gameTime * 60 || playersDead )
15        {
16            this.isReadyForNextStage = true;
17            this.timer = 0;
18            this.playersDead = false;
19        }
20        else
21        {
22            if (gameServer.checkIfPlayerDead ())
23            {
24                this.playersDead = true;
25                gameServer.enableMovement = false;
26            }
27            gameServer.enableMovement = true;
28            this.timer++;
29        }
30
31    }

```

32

33 }

```
1 package server;
2
3 public class StateChangingMap extends GameState
4 {
5     private int timer = -1;
6
7     @Override
8     public void handleUpdate ()
9     {
10         GameServer gameServer = GameServer.getInstance();
11         gameServer.enableMovement = false;
12         this.isReadyForNextStage = false;
13         if (this.timer == -1)
14         {
15             this.timer = 300;
16         }
17         gameServer.network.sendString("Winner player: "+gameServer.getWinner() + " map refresh in ... " + (this.timer / 60 + 1));
18         if (timer == 0)
19         {
20             gameServer.network.sendString(null);
21             this.isReadyForNextStage = true;
22             this.timer = -1;
23             gameServer.setGameLevel();
24         }
25         else
```

```

26     {
27         this.timer--;
28     }
29
30 }
31
32 }

```

```

1 package server;
2
3 public class StateWaitForPlayers extends GameState
4 {
5     private int timer = -1;
6
7     @Override
8     public void handleUpdate ()
9     {
10         GameServer gameServer = GameServer.getInstance();
11         this.isReadyForNextStage = false;
12         if (gameServer.getConnectedPlayerCount() > 2)
13         {
14             if (this.timer < 0)
15             {
16                 this.timer = 300;
17             }
18             gameServer.network.sendString("Game is starting in.. " + (this.
19 timer / 60 + 1));
20             if (timer == 0)
21             {

```

```

21     gameServer . network . sendString ( null );
22     this . isReadyForNextStage = true ;
23     this . timer = -1;
24 }
25 else
26 {
27     this . timer --;
28 }
29 }
30 else
31 {
32     this . timer = -1;
33     gameServer . network . sendString ( "Waiting for players.." );
34 }
35
36 }
37
38 }

```

```

1 package server;
2
3 import java . util . ArrayList ;
4 import java . util . List ;
5
6 import shared . Vector2f;
7
8 public class StateEndingGame extends GameState
9 {
10     private int timer = 0;

```

```

11 private boolean playersDead = false;
12 private int row = 1;
13 private int column = 1;
14 private int row2 = 1;
15 private int column2 = 1;
16 private int center = 0;
17 @Override
18 public void handleUpdate ()
19 {
20     GameServer gameServer = GameServer.getInstance();
21     this.isReadyForNextStage = false;
22     if (playersDead)
23     {
24         this.isReadyForNextStage = true;
25         this.timer = 0;
26         this.playersDead = false;
27         row = 0;
28         column = 1;
29         row2 = 1;
30         column2 = 1;
31         center = 0;
32     }
33     else
34     {
35         timer ++;
36         if (gameServer.checkIfPlayerDead())
37         {
38             this.playersDead = true;
39             gameServer.enableMovement = false;

```

```

40     }
41     gameServer.enableMovement = true ;
42     placeDeadlyBricks ();
43 }
44
45 }
46
47 private void placeDeadlyBricks ()
48 {
49     GameServer gameServer = GameServer.getInstance ();
50     if (timer % 15 == 0)
51     {
52         if (row < gameServer.gameBoard.gridSize - 2 - center)
53         {
54             gameServer.gameBoard.addObject(row, column, "wall");
55             this.damagePlayers(new Vector2f(row, column));
56             row ++;
57         }
58         else if(column < gameServer.gameBoard.gridSize - 1 - center)
59         {
60             gameServer.gameBoard.addObject(row, column, "wall");
61             this.damagePlayers(new Vector2f(row, column));
62             column ++;
63         }
64         else if (row2 < gameServer.gameBoard.gridSize - 2 - center)
65         {
66             gameServer.gameBoard.addObject(gameServer.gameBoard.gridSize -
row2 - 2, column - 1, "wall");
67             this.damagePlayers(new Vector2f(gameServer.gameBoard.gridSize -

```

```

        row2 - 2, column - 1));
68         row2 ++;
69     }
70     else if(column2 < gameServer.gameBoard.gridSize - 4 - center)
71     {
72         gameServer.gameBoard.addObject(gameServer.gameBoard.gridSize -
row2 - 1, gameServer.gameBoard.gridSize - column2 - 2, "wall");
73         this.damagePlayers(new Vector2f(gameServer.gameBoard.gridSize -
row2 - 1, gameServer.gameBoard.gridSize - column2 - 2));
74         column2 ++;
75     }
76     else
77     {
78         center ++;
79         row = center ;
80         column = center + 1;
81         row2 = center + 1;
82         column2 = center + 1;
83     }
84 }
85 }
86
87 private void damagePlayers(Vector2f bLoc){
88
89     GameServer session = GameServer.getInstance();
90
91     int cellSize = session.gameBoard.cellSize();
92
93     System.out.println("Kreipiuosi");

```

```

94
95
96     int kk = 0;
97
98     for(MPPlayer p : session.players.values())
99     {
100         if(p.coordinate != null){
101
102             if(kk != 0){
103
104                 List<Vector2f> pLoc = new ArrayList<>();
105
106                 int x1 = (int)p.coordinate.x / cellSize;
107                 int y1 = (int)p.coordinate.y / cellSize;
108
109                 int x2 = (int)(p.coordinate.x + p.size) / cellSize
110 ;
111
112                 int y2 = (int)p.coordinate.y / cellSize;
113
114                 int x3 = (int)p.coordinate.x / cellSize;
115                 int y3 = (int)(p.coordinate.y + p.size) / cellSize
116 ;
117
118                 int x4 = (int)(p.coordinate.x + p.size) / cellSize
119 ;
120                 int y4 = (int)(p.coordinate.y + p.size) / cellSize
121 ;
122
123                 Vector2f xy1 = new Vector2f(x1, y1);

```



```

119         Vector2f xy2 = new Vector2f(x2, y2);
120         Vector2f xy3 = new Vector2f(x3, y3);
121         Vector2f xy4 = new Vector2f(x4, y4);
122
123
124         pLoc.add(xy1);
125
126         if (!xy1.isEqual(xy2)){
127             pLoc.add(xy2);
128             System.out.println("Pridedu 2");
129         }
130
131         if (!xy1.isEqual(xy3) && !xy2.isEqual(xy3)){
132             pLoc.add(xy3);
133             System.out.println("Pridedu 3");
134         }
135
136         if (!xy1.isEqual(xy4) && !xy2.isEqual(xy4) && !xy3.
isEqual(xy4)){
137             pLoc.add(xy4);
138             System.out.println("Pridedu 4");
139         }
140
141
142         DamgePlayer(p, pLoc, bLoc);
143
144     }
145
146     kk++;

```

```

147
148     }
149 }
150
151 }
152
153 private void DamagePlayer(MPPlayer p, List<Vector2f> pLoc, Vector2f
bLoc){
154
155     System.out.println("Bombos: " + bLoc.x + " " + bLoc.y);
156
157     for(Vector2f Pxy : pLoc)
158     {
159         if (bLoc.isEqual (Pxy) )
160         {
161             System.out.println("Crushed by wall");
162             p.health = 0;
163             return;
164         }
165     }
166
167 }
168
169 }

```

```

1 package server;
2
3
4 import java.io.BufferedReader;

```

```
5 import java . io . IOException ;
6 import java . io . InputStreamReader ;
7 import java . util . Date ;
8 import java . util . HashMap ;
9 import java . util . Map ;
10 import java . util . Timer ;
11 import java . util . TimerTask ;
12
13 import shared . Vector2f ;
14 import shared . PacketUpdatePlayerPos ;
15
16 class GameServer
17 {
18     private static GameServer gameServer = null ;
19     //
20     //     this . stage1builder = new Stage1Builder ( gridSize ) ;
21     //     this . stage2builder = new Stage2Builder ( gridSize ) ;
22     //     this . stage3builder = new Stage3Builder ( gridSize ) ;
23     //
24
25
26     protected volatile GameBoard gameBoard ;
27
28     protected GameState currentState ;
29
30     protected volatile Map<Integer , MPPlayer> players ;
31     protected volatile Network network ;
32     private GameCycleThread thread ;
33     private Stage1Factory stage1factory ;
```

```
34
35 private int gameLevel ;
36 private int counter ;
37 private boolean readyForNextFrame ;
38 private Chain chain1 ;
39 private Chain chain2 ;
40 private Chain chain3 ;
41 private Chain chain4 ;
42 public boolean enableMovement = false ;
43
44 public ItemComponent powerUp ;
45
46 public ItemComponent everyItem ;
47
48 public Item dash ;
49 public Item teleport ;
50 public Item jump ;
51 public Item slowDown ;
52 public Item past ;
53
54 public Item bomb ;
55
56
57 public Item wall ;
58
59
60
61 public ItemList items ;
62
```

```
63  
64 private int levelChangeCounter ;
```

```
65  
66  
67 private boolean updateBoard = false ;
```

```
68  
69  
70 private GameServer ()
```

```
71 {
```

```
72  
73     powerUp = new ItemGroup("Powerup", "Folder that has all powerups  
that have been used");
```

```
74  
75     everyItem = new ItemGroup("Item list", "This is the list that hold  
data of PowerUps, Walls and Bombs");
```

```
76  
77  
78  
79     dash = new Item("Dash", "Player moves fast. ", 0);  
80     teleport = new Item("Teleport", "Player teleports. ", 0);  
81     slowDown = new Item("SlowDown", "Enemy get slowed down. ", 0);  
82     jump = new Item("Jump", "Player is capable of jumping over walls. "  
, 0);
```

```
83     past = new Item("Past", "Player is time traveler. ", 0);
```

```
84  
85  
86     bomb = new Item("Bomb", "Dangerous item. ", 0);
```

```
87     wall = new Item("Wall", "Blocks players path. ", 0);  
88
```

89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117

```
every Item . add ( powerUp ) ;

powerUp . add ( dash ) ;
powerUp . add ( tele port ) ;
powerUp . add ( slowDown ) ;
powerUp . add ( jump ) ;
powerUp . add ( past ) ;

every Item . add ( bomb ) ;
every Item . add ( wall ) ;

items = new Item List ( every Item ) ;

counter = 0;

levelChangeCounter = 0;

//Init Connection
if ( ! initConnection () )
{
    System . err . println ( "ERROR Connecting to host " );
    return ;
}
```

```
118     gameLevel = 2;
119
120     this.players = new HashMap<Integer , MPPlayer>();
121
122     setGameLevel ();
123
124     this.thread = new GameCycleThread ();
125     this.thread.start ();
126
127     chain1 = new GenerateWalls ();
128     chain3 = new GenerateDesWall ();
129     chain2 = new GenerateTrap ();
130     readyForNextFrame = true;
131     chain1.setNextChain ( chain2 );
132     chain2.setNextChain ( chain3 );
133
134     GameState waitForPlayersState = new StateWaitForPlayers ();
135     GameState playGameState = new StatePlayGame ();
136     GameState endingGameState = new StateEndingGame ();
137     GameState changingMapState = new StateChangingMap ();
138
139     waitForPlayersState.setNextState ( playGameState );
140     playGameState.setNextState ( endingGameState );
141     endingGameState.setNextState ( changingMapState );
142     changingMapState.setNextState ( waitForPlayersState );
143     currentState = waitForPlayersState;
144
145
146 }
```

```

147
148     protected GameServer getContext ()
149     {
150         return this;
151     }
152
153     public boolean checkIfPlayerDead () {
154         for (MPPlayer p : players.values ())
155         {
156             if (p.coordinate != null) {
157                 if (p.health < 1 && p.canDie) {
158                     p.deathCounter += 1;
159                     System.out.println ("Mires ");
160                     //setGameLevel ();
161                     return true;
162                 }
163             }
164         }
165         return false;
166     }
167
168
169     public String getWinner () {
170         boolean firstCycle = true;
171         for (MPPlayer p : players.values ())
172         {
173             if (!firstCycle)
174             {
175                 if (p.coordinate != null) {

```



```

176         if(p.health > 0){
177             //setGameLevel();
178             return String.valueOf(p.c.getID());
179         }
180     }
181 }
182 firstCycle = false;
183 }
184 return "None";
185
186 }
187
188 public int getConnectedPlayerCount(){
189     return players.size();
190
191 }
192 //nustatome sekancia busena
193 public void setState(GameState nextState)
194 {
195     this.currentState = nextState;
196 }
197
198
199 public void setGameLevel(){
200
201
202
203     if(gameLevel == 3)
204         gameLevel = 1;

```

```

205     else
206         gameLevel++;
207
208
209     for(MPPlayer p : players.values())
210     {
211         if(p.coordinate != null){
212             System.out.println("Atstatau gyvyvbes");
213             p.health = p.baseHealth ;
214             p.speed = p.baseSpeed ;
215
216         }
217     }
218
219
220
221     ChangeLevel ( gameLevel ) ;
222
223 }
224 public void stopPlayers()
225 {
226     for(MPPlayer p : players.values())
227     {
228         if(p.coordinate != null){
229             p.isHoldingDown = false ;
230             p.isHoldingUp = false ;
231             p.isHoldingLeft = false ;
232             p.isHoldingRight = false ;
233             p.isHoldingSkill = false ;

```

```
234     p.isHoldingUse = false ;
235 }
236
237
238 }
239 }
240
241 public void ChangeLevel(int level){
242
243     levelChangeCounter = 60 * 2;
244
245     stopPlayers();
246     gameLevel = level;
247
248     IStageBuilder builder;
249     AbstractFactory factory;
250     switch (level){
251         case 1:
252             default:
253                 builder = new Stage1Builder(20);
254                 factory = new Stage1Factory();
255                 this.gameBoard = new GameBoard(factory, builder, 1);
256                 break;
257         case 2:
258             builder = new Stage2Builder(20);
259             factory = new Stage1Factory();
260             this.gameBoard = new GameBoard(factory, builder, 2);
261             break;
262         case 3:
```

```
263     builder = new Stage3Builder (20);
264     factory = new Stage1Factory ();
265     this.gameBoard = new GameBoard( factory ,  builder ,  3);
266     break;
267 }
268 respawnAllPlayers ();
269 refreshMapForAllPlayers ();
270
271
272
273 }
274
275 public static GameServer getInstance()
276 {
277     if (gameServer == null)
278     {
279         gameServer = new GameServer ();
280     }
281     return gameServer ;
282 }
283
284 private boolean initConnection()
285 {
286     this.network = new Network ();
287
288     if (!this.network.initKryoServer())
289     {
290         return false ;
291     }
```

```

292
293     return true;
294 }
295
296 public void respawnAllPlayers ()
297 {
298     for(MPPlayer p : players.values())
299     {
300         if(p.coordinate != null){
301             this.respawnPlayer(p);
302         }
303
304
305     }
306
307 }
308
309 public void respawnPlayer (PlayerInfo p)
310 {
311     GameServer gameserver = GameServer.getInstance();
312     boolean teleported = false;
313     int maxRetry = 60;
314     int retry = 0;
315     while (!teleported && retry < maxRetry)
316     {
317         int randomCoordX = getRandomNumber (0, gameserver.gameBoard.
gridSize);
318         int randomCoordY = getRandomNumber (0, gameserver.gameBoard.
gridSize);

```

```

319         if (gameserver.gameBoard.objects[randomCoordX][randomCoordY]
instanceof Ground)
320         {
321             p.coordinate.x = randomCoordX * (gameserver.gameBoard.size /
gameserver.gameBoard.gridSize);
322             p.coordinate.y = randomCoordY * (gameserver.gameBoard.size /
gameserver.gameBoard.gridSize);
323             teleported = true;
324         }
325         retry++;
326     }
327 }
328
329 private int getRandomNumber(int min, int max)
330 {
331     return (int) ((Math.random() * (max - min)) + min);
332 }
333
334 public void updatePlayer(int id, PacketUpdatePlayerPos playerPacket)
335 {
336     MPPlayer player = players.get(id);
337     if (player != null && levelChangeCounter < 0)
338     {
339         player.isHoldingUp = playerPacket.isHoldingUp != null ?
playerPacket.isHoldingUp : player.isHoldingUp;
340         player.isHoldingDown = playerPacket.isHoldingDown != null ?
playerPacket.isHoldingDown : player.isHoldingDown;
341         player.isHoldingLeft = playerPacket.isHoldingLeft != null ?
playerPacket.isHoldingLeft : player.isHoldingLeft;

```

```
342     player.isHoldingRight = playerPacket.isHoldingRight != null ?
playerPacket.isHoldingRight : player.isHoldingRight ;
343     player.isHoldingUse = playerPacket.isHoldingUse != null ?
playerPacket.isHoldingUse : player.isHoldingUse ;
344     player.isHoldingSkill = playerPacket.isHoldingSkill != null ?
playerPacket.isHoldingSkill : player.isHoldingSkill ;
345     players.put(player.id , player);
346 }
347 }
348
349 public void addPlayer(MPPlayer player)
350 {
351     this.players.put(player.c.getID() , player);
352     this.respawnPlayer(player);
353     this.network.sendGameBoard(gameBoard , player);
354
355 }
356
357 public void refreshMapForAllPlayers()
358 {
359     for(MPPlayer p : players.values())
360     {
361         network.sendPlayerInfo(p , true);
362         this.network.sendGameBoard(gameBoard , p);
363     }
364 }
365
366 public void removePlayer(int id)
367 {
```

```

368
369     players.remove(id);
370 }
371
372
373 private class GameCycleThread extends Thread
374 {
375     volatile boolean isGameRunning = true;
376     private final int gameSpeed = 16; //The lower the number the
377                                         faster the game is
378
379
380     public GameCycleThread ()
381     {
382         this.isGameRunning = true;
383     }
384
385     public void run ()
386     {
387         long lastTime = System.nanoTime();
388         final double ns = 1000000000.0 / 60.0;
389         double delta = 0;
390         while (true){
391             long now = System.nanoTime();
392             delta += (now - lastTime) / ns;
393             lastTime = now;
394             while (delta >= 1){
395                 update();

```



```

396         delta --;
397     }
398 }
399 }
400
401 public void stopGame()
402 {
403     this.isGameRunning = false;
404
405     //network should probably be closed by the parent
406     network.close();
407 }
408
409 private void update()
410 {
411
412     counter++;
413     levelChangeCounter --;
414     currentState.handleUpdate();
415     if (currentState.isReadyForNextStage())
416     {
417         currentState.getNextState(getContext());
418     }
419
420     if (enableMovement)
421     {
422         updatePlayers();
423         gameBoard.runTick();
424     }

```

```
425     surenkamTeksta ();
426
427
428     checkIfPlayerDead ();
429
430
431
432
433     }
434
435
436
437     private void surenkamTeksta () {
438         InputStreamReader fileInputStream=new InputStreamReader (System . in
439 );
440         BufferedReader bufferedReader=new BufferedReader (fileInputStream)
441 ;
442
443         String tekstas = "nieko nenuskaityta ";
444
445         try {
446             if (bufferedReader . ready ()) {
447
448                 tekstas = bufferedReader . readLine ();
449
450
451                 System . out . println ("irasytas tekstas " + tekstas);
```

```

452
453
454
455     String tekstasLower = tekstas.toLowerCase();
456
457
458     String[] sarasas = tekstasLower.split(" ");
459
460
461     if(sarasas.length == 1){
462
463         items.getItemList();
464
465     }
466     else if(sarasas.length > 3){
467         ConversionContext task = new ConversionContext(tekstas);
468
469         String whatToPlace = task.getWhat();
470         String whereToPlace = task.getWhere();
471         int howManyToPlace = task.getQuantity();
472
473
474         Task newTask = new Task(howManyToPlace, whatToPlace,
whereToPlace, gameBoard);
475
476         gameBoard = chain1.result(newTask);
477
478         System.out.println(whatToPlace + " How many: " +
howManyToPlace);

```

```

479     } else if (sarasas.length > 1){
480
481         if (sarasas[0].equals("change") && sarasas[1].equals("level"
482     )){
483         int lygis;
484         try {
485             lygis = Integer.parseInt(sarasas[2]);
486
487             if (lygis > 0 && lygis < 4){
488                 ChangeLevel(lygis);
489
490             } else {
491                 System.out.println("Toks lygis neegzistuoja: " +
492     lygis);
493             }
494
495             } catch (NumberFormatException e) {
496                 System.out.println("Toks lygis neegzistuoja: " +
497     sarasas[2]);
498             }
499         }
500
501         else {
502             System.out.println("Unknown command " + tekstas);
503         }
504
505         System.out.println("Unknown command " + tekstas);

```

```
505         }
506
507
508
509     }
510
511
512     } catch (IOException e) {
513         e.printStackTrace();
514     }
515
516
517
518
519
520
521
522
523
524 }
525
526
527 private void updatePlayers ()
528 {
529     int k = 0;
530     for (MPPlayer p : players.values())
531     {
532         if (k!= 0)
533         {
```

```

534         if (p.isHoldingSkill)
535         {
536             p.tryUsingSpell();
537         }
538
539         p.onTick();
540         p.reduceTimer();
541
542         if (p.isHoldingUse)
543         {
544
545             if (p.bombTimer < 0){
546                 gameBoard.SpawnBomb(p);
547
548                 p.setBombTimer(2);
549             }
550
551         }
552         p.coordinate = checkCollision(p);
553         network.sendGameBoard(gameBoard, p);
554         network.sendPlayerInfo(p, true);
555     }
556     k++;
557 }
558 }
559
560 private Vector2f checkCollision(MPPlayer p)
561 {
562

```

```

563     Vector2f coordsAfterMove = new Vector2f(p.coordinate.x, p.
coordinate.y);

564

565     float padding = 0.001f;
566     float cellSize = gameBoard.cellSize();
567
568     boolean moveX = true;
569     boolean moveY = true;
570
571     if (p.isHoldingLeft)
572     {
573         coordsAfterMove.x -= p.speed;
574     }
575
576     if (p.isHoldingRight)
577     {
578
579         coordsAfterMove.x += p.speed;
580     }
581
582     boolean collidingLeft = ((int)coordsAfterMove.x / cellSize -
padding) < ((int)p.coordinate.x / cellSize);
583     boolean collidingRight = (((int)coordsAfterMove.x + p.size +
padding) / cellSize) > (((int)p.coordinate.x + p.size) / cellSize);
584     boolean isCollidingX = collidingLeft || collidingRight;
585     moveX = !(coordsAfterMove.x <= 0 || coordsAfterMove.x >=
gameBoard.size - p.size);
586     //Some smoothing when going around edges would be nice
587     if (isCollidingX && moveX)

```

```

588 {
589     int x = 0, y = 0, y1 = 0;
590     if (collidingLeft)
591     {
592         x = (int) ((coordsAfterMove.x) / cellSize);
593         y = (int) (p.coordinate.y / cellSize);
594         y1 = (int) ((p.coordinate.y + p.size) / cellSize);
595
596     }
597     if (collidingRight)
598     {
599         x = (int) ((coordsAfterMove.x + p.size) / cellSize);
600         y = (int) (p.coordinate.y / cellSize);
601         y1 = (int) ((p.coordinate.y + p.size) / cellSize);
602
603     }
604
605     if (y == y1)
606     {
607         moveX = gameBoard.objects[x][y].isWalkable;
608         gameBoard.objects[x][y].onStep(p);
609     }
610     else
611     {
612         moveX = gameBoard.objects[x][y].isWalkable && gameBoard.
objects[x][y1].isWalkable;
613         gameBoard.objects[x][y].onStep(p);
614         gameBoard.objects[x][y1].onStep(p);
615     }

```



```

616     }
617
618
619     if (p.isHoldingUp)
620     {
621         coordsAfterMove.y += p.speed;
622     }
623
624     if (p.isHoldingDown)
625     {
626         coordsAfterMove.y -= p.speed;
627     }
628
629     boolean collidingUp = (((int)coordsAfterMove.y + p.size - padding
630 ) / cellSize) > (((int)p.coordinate.y + p.size) / cellSize);
631     boolean collidingDown = ((int)coordsAfterMove.y / cellSize +
632 padding) < ((int)p.coordinate.y / cellSize);
633
634     boolean isCollidingY = collidingUp || collidingDown;
635     moveY = !(coordsAfterMove.y <= 0 || coordsAfterMove.y >=
636 gameBoard.size - p.size);
637
638
639     //Some smoothing when going around edges would be nice
640     if (isCollidingY && moveY)
641     {
642         int x = 0, x1 = 0, y = 0;
643
644         if (collidingUp)
645         {
646             y = (int) ((coordsAfterMove.y + p.size) / cellSize);

```

```

642         x = (int) (p.coordinate.x / cellSize);
643         x1 = (int) ((p.coordinate.x + p.size) / cellSize);
644
645     }
646     if (collidingDown)
647     {
648         y = (int) ((coordsAfterMove.y) / cellSize);
649         x = (int) (p.coordinate.x / cellSize);
650         x1 = (int) ((p.coordinate.x + p.size) / cellSize);
651     }
652
653     if (x == x1)
654     {
655         moveY = gameBoard.objects[x][y].isWalkable;
656         gameBoard.objects[x][y].onStep(p);
657     }
658     else
659     {
660         moveY = gameBoard.objects[x][y].isWalkable && gameBoard.
objects[x1][y].isWalkable;
661         gameBoard.objects[x][y].onStep(p);
662         gameBoard.objects[x1][y].onStep(p);
663     }
664 }
665
666 coordsAfterMove.x = moveX ? coordsAfterMove.x : p.coordinate.x;
667 coordsAfterMove.y = moveY ? coordsAfterMove.y : p.coordinate.y;
668
669 return coordsAfterMove;

```

670

671

}

672

673

}

674

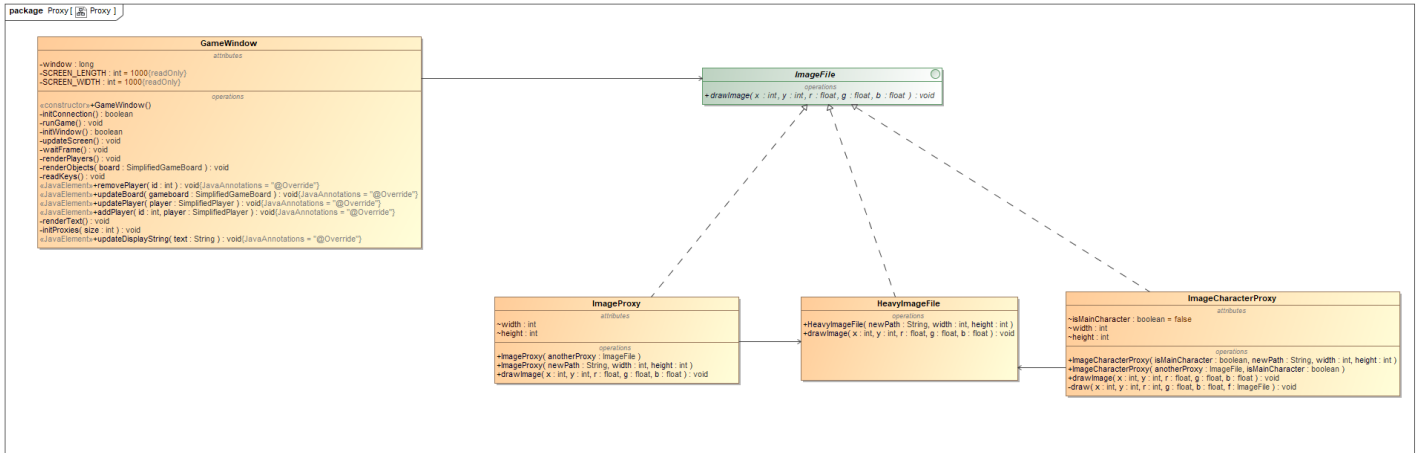
675

}

Proxy šablonas

Kuriant projektą prirėikė efektyvaus būdo apdoroti nuotraukas, kad nereikėtų į atmintį iš naujo įkėlinėti ir naudoti tik tada kada reikia. Taip pat kilo noras tekstūras šiek tiek modifikuoti jau įkėlus į atmintį. Šiuos pageidavimus tikslingai padeda įvykdyti Proxy šablonas

UML diagrama:



Greitaveika lyginant naudojimą be šio šablono:

```
=== Single Image ===
One image loaded in: 3487 ms

=== Heavy Images == - show one image only ===
Real loaded in 8299 ms

=== Proxy Images - show one image only ===
Proxy loaded in 2810ms

=== Proxy Images - run all ===
Proxy loaded in 8344ms
|
```

Kaip matome skirtumas labia didelis

Kodas

```
package client;

1 public class ImageCharacterProxy implements ImageFile {
2
3     ImageFile file;
4
5     String path;
6
7     boolean isMainCharacter = false;
8     String user = null;
9     int width;
10    int height;
11    ImageFile parentProxy;
12
13    public ImageCharacterProxy(boolean isMainCharacter, String newPath,
14        int width, int height)
15    {
16        this.isMainCharacter = isMainCharacter;
17        this.path = newPath;
18        this.width = width;
19        this.height = height;
20    }
```

```

21 }
22
23 public ImageCharacterProxy (ImageFile anotherProxy , boolean
    isMainCharacter ){
24     parentProxy = anotherProxy ;
25     this.isMainCharacter = isMainCharacter ;
26 }
27
28
29 @Override
30 public void drawImage(int x, int y, float r, float g, float b)
31 {
32     if(parentProxy == null)
33     {
34         if(file == null)
35         {
36             file = new HeavyImageFile (this.path, this.width, this.height);
37             this.draw(x, y, y, g, b, file);
38         }
39         else
40         {
41             this.draw(x, y, y, g, b, file);
42         }
43     }
44 }
45 else
46 {
47     this.draw(x, y, y, g, b, parentProxy );
48 }

```

```

49     }
50
51     private void draw(int x, int y, int r, float g, float b, ImageFile f)
52     {
53
54         if(isMainCharacter)
55         {
56             //Removing color for main char
57             f.drawImage(x, y, 1f, 1f, 1f);
58         }
59         else
60         {
61             f.drawImage(x, y, r, g, b);
62             float size = 40;
63             float xEn = x*4/size;
64             float yEn = y*4/size;
65             Text.drawString("enemy",xEn- 8/10 ,yEn-1 , size , 2);
66         }
67     }
68 }

```

```

1 package client;
2
3 import java.awt.image.BufferedImage;
4 import java.io.File;
5 import java.io.IOException;
6
7 import javax.imageio.ImageIO;
8

```

```

9
10 public class HeavyImageFile implements ImageFile {
11
12     Sprite heavySprite ;
13     String path ;
14     private TextureLoader textureLoader ;
15
16     public HeavyImageFile (String newPath, int width, int height)
17     {
18         textureLoader = new TextureLoader () ;
19         this.heavySprite = new Sprite (textureLoader , newPath, width, height
20         );
21     }
22
23     @Override
24     public void drawImage (int x, int y, float r, float g, float b) {
25         this.heavySprite.draw (x, y, r, g, b);
26     }
27
28 }

```

```

1 package client;
2
3 public class ImageProxy implements ImageFile {
4
5     HeavyImageFile file = null; //heavy image file
6     int width ;
7     int height ;

```



```

8  //Heavy ImageFile  file ;
9
10 String path ;
11 ImageFile  parentProxy ;
12
13 public ImageProxy (String newPath, int width , int height )
14 {
15     this .path = newPath ;
16     this .width = width ;
17     this .height = height ;
18 }
19
20 public ImageProxy (ImageFile  anotherProxy ){
21     parentProxy = anotherProxy ;
22 }
23
24 @Override
25 public void drawImage (int x, int y, float r, float g, float b)
26 {
27     if (parentProxy == null)
28     {
29         if (file == null)
30         {
31             file = new HeavyImageFile (path , width , height );
32         }
33         file .drawImage (x, y, r, g, b);
34     }
35     else
36     {

```

```
37     parentProxy.drawImage(x, y, r, g, b);
38 }
39
40 }
41
42
43 }
```

```
1 package client;
2
3
4 public interface ImageFile
5 {
6
7     public void drawImage(int x, int y, float r, float g, float b);
8
9 }
```

```
1 package client;
2
3 import java . nio . ByteBuffer ;
4 import java . util . HashMap ;
5 import java . util . Map ;
6 import org . lwjgl . glfw . GLFW ;
7 import org . lwjgl . opengl . GL11 ;
8
9 import shared . SimplifiedGameBoard ;
10 import shared . SimplifiedPlayer ;
11 import shared . Vector2f ;
12 import shared . ObjectType ;
```

```

13
14 import org.lwjgl.opengl.GL;
15
16
17
18 public class GameWindow implements UpdateGameDataDelegate
19 {
20     private long window;
21     final private int SCREEN_LENGTH = 1000;
22     final private int SCREEN_WIDTH = 1000;
23     private SimplifiedGameBoard board;
24     private Network network;
25     private SimplifiedPlayer mainPlayer;
26     private Map<Integer, ClientPlayer> players;
27     private String displayString = null;
28     //Textures
29     private ImageFile bombSprite;
30     private ImageFile wallSprite;
31     private ImageFile groundSprite;
32     private ImageFile powerupSprite;
33     private ImageFile trapSprite;
34     private ImageFile fireSprite;
35     private ImageFile playerSprite;
36
37     public GameWindow()
38     {
39
40         //Init Window
41         if (!initWindow())

```

```

42     {
43         System.err.println("ERROR WHILE INITIATING WINDOW");
44         return ;
45     }
46
47     //Init Connection
48     if (!initConnection())
49     {
50         System.err.println("ERROR Connecting to host");
51         return ;
52     }
53
54
55     this.mainPlayer = new SimplifiedPlayer(this.network.client.getID());
56     ;
57     this.players = new HashMap<Integer, Client Player>();
58     runGame();
59 }
60
61 private void initProxies(int size)
62 {
63     bombSprite = new ImageProxy("Tnt.png", size, size);
64     wallSprite = new ImageProxy("Cobble.png", size, size);
65     groundSprite = new ImageProxy("Grass.png", size, size);
66     powerupSprite = new ImageProxy("Powerup.png", size, size);
67     trapSprite = new ImageProxy("Trap.png", size, size);
68     fireSprite = new ImageProxy("Fire.png", size, size);
69     playerSprite = new ImageProxy("Player.jpg", 40, 40);

```

```
70 }
71
72 private boolean initConnection()
73 {
74     this.network = new Network();
75
76     if (!this.network.connect(this))
77     {
78         return false;
79     }
80
81     return true;
82 }
83
84 private void runGame()
85 {
86     while (GLFW.glfwWindowShouldClose(window) != true)
87     {
88         readKeys();
89         updateScreen();
90         waitFrame();
91     }
92 }
93
94 private boolean initWindow()
95 {
96     if (GLFW.glfwInit() != true)
97     {
98         return false;
```

```

99     }
100
101     window = GLFW.glfwCreateWindow (SCREEN_LENGTH, SCREEN_WIDTH, "Window
102
103     GLFW.glfwShowWindow ( window ) ;
104     GLFW.glfwMakeContextCurrent ( window ) ;
105     GL.createCapabilities () ;
106
107     //Init GL
108     // enable textures since we're going to use these for our sprites
109     GL11.glEnable (GL11.GL_TEXTURE_2D) ;
110     GL11.glEnable (GL11.GL_BLEND) ;
111     GL11.glDepthFunc (GL11.GL_ALWAYS) ;
112     GL11.glBlendFunc (GL11.GL_SRC_ALPHA, GL11.GL_ONE_MINUS_SRC_ALPHA) ;
113     // disable the OpenGL depth test since we're rendering 2D graphics
114     GL11.glEnable (GL11.GL_DEPTH_TEST) ;
115     GL11.glMatrixMode (GL11.GL_PROJECTION) ;
116     GL11.glLoadIdentity () ;
117     GL11.glOrtho (0, SCREEN_LENGTH, 0, SCREEN_WIDTH, -1, 1) ;
118     GL11.glMatrixMode (GL11.GL_MODELVIEW) ;
119     return true ;
120 }
121 private void updateScreen ()
122 {
123     GLFW.glfwPollEvents () ;
124     GL11.glClear (GL11.GL_COLOR_BUFFER_BIT | GL11.GL_DEPTH_BUFFER_BIT) ;
125     GL11.glEnable (GL11.GL_TEXTURE_2D) ;
126     renderObjects ( board ) ;

```

```

127     renderPlayers ();
128     GL11.glDisable (GL11.GL_TEXTURE_2D);
129     renderText ();
130     GLFW.glfwSwapBuffers (window );
131 }
132
133 private void renderText ()
134 {
135     Text.drawString ("Spell " + this.mainPlayer.skillName + " Cooldown
" + (int)(this.mainPlayer.skillCooldown / 60), 5, 2, 40, 2);
136     Text.drawString ("Player " + this.mainPlayer.id + " Health " + this.
mainPlayer.health , 45, 2, 40, 2);
137
138     int size = 40;
139
140
141
142     if (this.mainPlayer.health < 1){
143
144         Text.drawString ("GG", 2, 6, 100, 20);
145     }
146
147     for (SimplifiedPlayer mpPlayer : players.values())
148     {
149
150         if (this.mainPlayer.id != mpPlayer.id){
151
152             if (mpPlayer.coordinate != null)
153             {

```

```

154         float x = mpPlayer.coordinate.x*4/size;
155         float y = mpPlayer.coordinate.y*4/size;
156
157     }
158
159 }
160
161 }
162
163 if (this.displayString != null)
164 {
165     Text.drawString(this.displayString, 10, 40, 55, 4);
166 }
167
168
169 }
170
171 //bad design
172 private void waitFrame()
173 {
174     try
175     {
176         //around 120 tiems a second
177         Thread.sleep(1L);
178     }
179     catch (InterruptedException e)
180     {
181         // TODO Auto-generated catch block
182         e.printStackTrace();

```



```

183     }
184 }
185 private void renderPlayers()
186 {
187     for(ClientPlayer mpPlayer : players.values()){
188         mpPlayer.getSprite().drawImage((int)mpPlayer.coordinate.x, (int)
189         mpPlayer.coordinate.y, 0f, (float)(255/255), (float)(127/255));
190     }
191 }
192
193 private void renderObjects (SimplifiedGameBoard board)
194 {
195     GL11.glEnable (GL11.GL_BLEND);
196     GL11.glBlendFunc (GL11.GL_SRC_ALPHA, GL11.GL_ONE_MINUS_SRC_ALPHA);
197     if (board == null)
198     {
199         return;
200     }
201     int sizeX = SCREEN_LENGTH / board.gridSize;
202     int sizeY = SCREEN_WIDTH / board.gridSize;
203
204
205     for (int i = 0; i < board.gridSize; i++) {
206         for (int j = 0; j < board.gridSize; j++) {
207             float red = (float)Integer.valueOf(board.objects[i][j].color.
208             substring(1,3), 16)/255;
209             float green = (float)Integer.valueOf(board.objects[i][j].color.
210             substring(3,5), 16)/255;

```

```

209         float blue = (float)Integer.valueOf(board.objects[i][j].color.
substring(5,7), 16)/255;
210         ImageProxy drawer;
211         switch (board.objects[i][j].type)
212         {
213             default:
214             case GROUND:
215                 drawer = new ImageProxy (groundSprite);
216                 break;
217             case WALL:
218                 drawer = new ImageProxy (wallSprite);
219                 break;
220             case TRAP:
221                 drawer = new ImageProxy (trapSprite);
222                 break;
223             case POWERUP:
224                 drawer = new ImageProxy (powerupSprite);
225                 break;
226             case BOMB:
227                 drawer = new ImageProxy (bombSprite);
228                 break;
229         }
230
231         drawer.drawImage(i*sizeX, j*sizeY, red, green, blue);
232         GL11.glDisable(GL11.GL_BLEND);
233         if (board.objects[i][j].explodeAnimation == true)
234         {
235             GL11.glBlendFunc(GL11.GL_SRC_ALPHA, GL11.
GL_ONE_MINUS_SRC_ALPHA);

```

```

236         drawer = new ImageProxy ( fireSprite );
237         drawer.drawImage ( i*sizeX , j*sizeY , 1, 1, 1);
238     }
239 }
240 }
241
242
243
244 }
245
246 private void readKeys ()
247 {
248
249     //TODO should read from events instead of polling
250     //https://www.glfw.org/docs/3.3/input_guide.html
251
252
253
254
255     if (GLFW.glfwGetKey ( this.window , GLFW.GLFW_KEY_W) == GLFW.GLFW_TRUE)
256     {
257         if (! this.mainPlayer.isHoldingUp )
258         {
259             this.network.sendPacketButtonPressUp () ;
260         }
261     }
262     else
263     {
264         if ( this.mainPlayer.isHoldingUp )

```

```
265     {
266         this.network.sendPacketButtonReleaseUp();
267     }
268
269 }
270
271 if (GLFW.glfwGetKey(this.window, GLFW.GLFW_KEY_S) == GLFW.GLFW_TRUE)
272 {
273     if (!this.mainPlayer.isHoldingDown)
274     {
275         this.network.sendPacketButtonPressDown();
276     }
277 }
278 else
279 {
280     if (this.mainPlayer.isHoldingDown)
281     {
282         this.network.sendPacketButtonReleaseDown();
283     }
284 }
285
286
287 if (GLFW.glfwGetKey(this.window, GLFW.GLFW_KEY_A) == GLFW.GLFW_TRUE)
288 {
289     if (!this.mainPlayer.isHoldingLeft)
290     {
291         this.network.sendPacketButtonPressLeft();
292     }
293 }
```

```
294     else
295     {
296         if ( this .mainPlayer .isHoldingLeft)
297         {
298             this .network . send Packet Button Release Left () ;
299         }
300
301     }
302
303     if (GLFW.glfwGetKey( this . window , GLFW.GLFW_KEY_D) == GLFW.GLFW_TRUE)
304     {
305         if ( ! this .mainPlayer .isHolding Right )
306         {
307             this .network . send Packet Button Press Right () ;
308         }
309     }
310     else
311     {
312         if ( this .mainPlayer .isHolding Right )
313         {
314             this .network . send Packet Button Release Right () ;
315         }
316
317     }
318
319
320     if (GLFW.glfwGetKey( this . window , GLFW.GLFW_KEY_E) == GLFW.GLFW_TRUE)
321     {
322         if ( ! this .mainPlayer .isHolding Use )
```

```

323     {
324         this.network.sendPacketButtonPressUse();
325     }
326 }
327 else
328 {
329     if (this.mainPlayer.isHoldingUse)
330     {
331         this.network.sendPacketButtonReleaseUse();
332     }
333 }
334
335 if(GLFW.glfwGetKey(this.window, GLFW.GLFW_KEY_SPACE) == GLFW.
GLFW_TRUE)
336 {
337     if (!this.mainPlayer.isHoldingSkill)
338     {
339         this.network.sendPacketButtonPressSkill();
340     }
341 }
342 else
343 {
344     if (this.mainPlayer.isHoldingSkill)
345     {
346         this.network.sendPacketButtonReleaseSkill();
347     }
348 }
349
350 if (GLFW.glfwGetKey(window, GLFW.GLFW_KEY_ESCAPE) == GLFW.GLFW_TRUE

```

```

    )
351 {
352     GLFW.glfwSetWindowShouldClose ( window ,  true );
353 }
354 }
355
356 @Override
357 public void  updatePlayer (SimplifiedPlayer  player)
358 {
359     if (this.mainPlayer.id == player.id)
360     {
361         ClientPlayer  clientPlayer = new  ClientPlayer(player ,  playerSprite
,  true );
362         this.mainPlayer = player;
363         this.players.put(player.id ,  clientPlayer);
364     }
365     else
366     {
367         ClientPlayer  clientPlayer = new  ClientPlayer(player ,  playerSprite
,  false );
368         this.players.put(player.id ,  clientPlayer);
369     }
370
371 }
372 @Override
373 public void  addPlayer (int  id ,  SimplifiedPlayer  player)
374 {
375     if (player.id == mainPlayer.id )
376     {

```

```

377     ClientPlayer clientPlayer = new ClientPlayer(player , playerSprite
, true);
378     players.put(id , clientPlayer);
379 }
380 else
381 {
382     ClientPlayer clientPlayer = new ClientPlayer(player , playerSprite
, false);
383     players.put(id , clientPlayer);
384 }
385 }
386
387 @Override
388 public void removePlayer (int id)
389 {
390     players.remove(id);
391 }
392
393 @Override
394 public void updateBoard (SimplifiedGameBoard gameboard)
395 {
396     if ( this.board == null)
397     {
398         this.initProxies(gameboard.size /gameboard.gridSize);
399     }
400     this.board = gameboard;
401
402 }
403

```



```
404
405 @Override
406 public void updateDisplayString (String text)
407 {
408     this.displayString = text;
409
410 }
411
412 }
```

```
1 package client;
2
3 import java .awt .image .BufferedImage ;
4 import java .io .File ;
5 import java .io .IOException ;
6 import java .util .HashMap ;
7
8 import javax .imageio .ImageIO ;
9
10 import org .lwjgl .glfw .GLFW ;
11 import org .lwjgl .opengl .GL ;
12 import org .lwjgl .opengl .GL11 ;
13
14 import shared .SimplifiedPlayer ;
15
16
17 public class ProxyTest {
18
19     public static void main (String [] args) {
```

```

20     if (GLFW.glfwInit() != true)
21     {
22         return ;
23     }
24
25     long window ;
26     int SCREEN_LENGTH = 1000;
27     int SCREEN_WIDTH = 1000;
28     //Init Window
29     window = GLFW.glfwCreateWindow(SCREEN_LENGTH, SCREEN_WIDTH, "Window
", 0, 0);
30
31     GLFW.glfwShowWindow ( window ) ;
32     GLFW.glfwMakeContextCurrent ( window ) ;
33     GL.createCapabilities();
34
35     //Init GL
36     // enable textures since we're going to use these for our sprites
37     GL11.glEnable(GL11.GL_TEXTURE_2D);
38     GL11.glEnable(GL11.GL_BLEND);
39     GL11.glDepthFunc(GL11.GL_ALWAYS);
40     GL11.glBlendFunc ( GL11.GL_SRC_ALPHA, GL11.GL_ONE_MINUS_SRC_ALPHA );
41     // disable the OpenGL depth test since we're rendering 2D graphics
42     GL11.glEnable(GL11.GL_DEPTH_TEST);
43     GL11.glMatrixMode(GL11.GL_PROJECTION);
44     GL11.glLoadIdentity();
45     GL11.glOrtho(0, SCREEN_LENGTH, 0, SCREEN_WIDTH, -1, 1);
46     GL11.glMatrixMode(GL11.GL_MODELVIEW);
47

```

```

48
49
50 String path = "Test.jpg";
51
52 System.out.println("== Single Image ==");
53 long startP = System.currentTimeMillis();
54 TextureLoader loader = new TextureLoader();
55 Sprite newsprite = new Sprite(loader, path, 50, 50);
56
57 long stopP = System.currentTimeMillis();
58
59 System.out.println("One image loaded in: " + (stopP - startP) + "
ms");
60
61 System.gc(); // Runtime.getRuntime().gc();
62
63 System.out.println("\n== Heavy Images == - show one image only
====");
64 long startReal = System.currentTimeMillis();
65 ImageFile h1 = new HeavyImageFile(path, 50, 50);
66 ImageFile h2 = new HeavyImageFile(path, 50, 50);
67 ImageFile h3 = new HeavyImageFile(path, 50, 50);
68 h2.drawImage(0, 0, 1, 1, 1);
69 long stopReal = System.currentTimeMillis();
70 System.out.println("Real loaded in " + (stopReal - startReal) + "
ms");
71
72 System.gc(); // Runtime.getRuntime().gc();
73

```

```

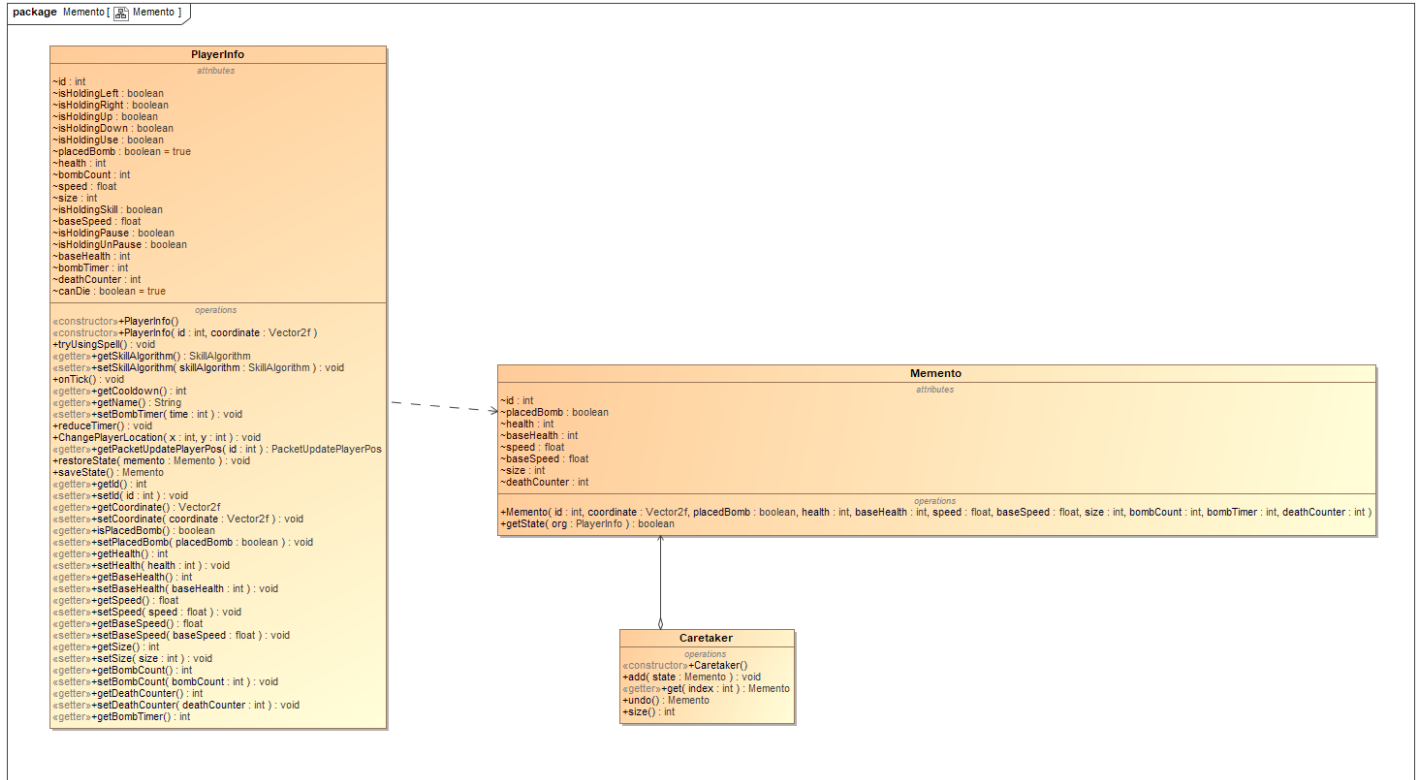
74 System.out.println("\n== Proxy Images - show one image only ==");
75 long startProxy = System.currentTimeMillis();
76 ImageFile p1 = new ImageProxy(path,50,50);
77 ImageFile p2 = new ImageProxy(path,50,50);
78 ImageFile p3 = new ImageProxy(path,50,50);
79 p2.drawImage(0, 0, 1,1,1);
80 long stopProxy = System.currentTimeMillis();
81 System.out.println("Proxy loaded in " + (stopProxy - start Proxy) +
"ms");
82
83 System.gc(); // Runtime.getRuntime().gc();
84
85 System.out.println("\n== Proxy Images - run all ==");
86 long startProxyA = System.currentTimeMillis();
87 ImageFile a1 = new ImageProxy(path,50,50);
88 ImageFile a2 = new ImageProxy(path,50,50);
89 ImageFile a3 = new ImageProxy(path,50,50);
90 a1.drawImage(0, 0, 1,1,1);
91 a2.drawImage(0, 0, 1,1,1);
92 a3.drawImage(0, 0, 1,1,1);
93 long stopProxyA = System.currentTimeMillis();
94 System.out.println("Proxy loaded in " + (stopProxyA - startProxyA)
+ "ms");
95 }
96
97 }

```

Memento

Kilo idėja į žaidimą įdėti įgūdį, kuris leidžia vartotojui “grįžti laiku” ir nekreipti dėmesio į klaidas, kurias padarė. Šiai idėjai tikslingai pravertė memento šablonas.

UML diagrama:



Kodas:

```
2
3 import shared . Vector2f;
4
5 public class Memento {
6
7     int id;
8     Vector2f coordinate;
9     boolean placedBomb;
10    int health;
11    int baseHealth;
12    float speed;
13    float baseSpeed;
14    int size;
15    int deathCounter;
16
17
18
19    public Memento( int id ,
20        Vector2f coordinate ,
21        boolean placedBomb, int health ,
22        int baseHealth , float speed , float baseSpeed ,
23        int size , int bombCount, int bombTimer, int deathCounter )
24    {
25        this.id = id;
26        this.coordinate = coordinate;
27        this.placedBomb = placedBomb;
28        this.health = health;
```

```
29     this.baseHealth = baseHealth ;
```

```
30     this.speed = speed ;
```

```

31     this.baseSpeed = baseSpeed ;
32     this.size = size ;
33     this.deathCounter = deathCounter ;
34 }
35
36 public boolean getState (PlayerInfo org) {
37     if(id == org.getId ())
38     {
39         org.setCoordinate (coordinate );
40         org.setPlacedBomb (placedBomb );
41         org.setHealth (health );
42         org.setBaseHealth (baseHealth );
43         org.setSpeed (baseSpeed );
44         org.setBaseSpeed (baseSpeed );
45         org.setSize (size );
46
47         return true ;
48     }
49     return false ;
50
51 }
52
53 }

```

```

1 package server;
2
3 import java.util.*;
4
5

```



```
6 public class Caretaker {
7
8     ArrayList<Memento> statesList;
9
10    public Caretaker(){
11        statesList = new ArrayList<Memento>();
12    }
13
14    public void add(Memento state){
15        statesList.add(state);
16    }
17
18    public Memento get(int index){
19        Memento restoreState = statesList.get(index);
20        statesList.remove(index);
21        return restoreState;
22    }
23
24    public Memento undo()
25    {
26        //popping last state
27        int index = statesList.size() -1;
28        Memento restoreState = statesList.get(index);
29        statesList.remove(index);
30        return restoreState;
31    }
32
33
34    public int size(){
```

```
35     return statesList.size();
36 }
37
38 }
```

```
1 package server;
2
3 import shared.PacketUpdatePlayerPos;
4 import shared.Vector2f;
5
6 import java.util.Random;
7
8 public class PlayerInfo
9 {
10     int id;
11     Vector2f coordinate;
12     boolean isHoldingLeft;
13     boolean isHoldingRight;
14     boolean isHoldingUp;
15     boolean isHoldingDown;
16     boolean isHoldingUse;
17     boolean isHoldingSkill;
18     boolean isHoldingPause;
19     boolean isHoldingUnPause;
20     boolean placedBomb = true;
21     int health;
22     int baseHealth;
23     float speed;
24     float baseSpeed;
```

```
25  int size ;
26  int bombCount ;
27  int bombTimer ;
28
29  int deathCounter ;
30
31  SkillAlgorithm skillAlgorithm ;
32
33  boolean canDie = true ;
34
35  public PlayerInfo ()
36  {
37
38      deathCounter = 0 ;
39
40      Random r = new Random () ;
41      int low = 100 ;
42      int high = 800 ;
43
44      setBombTimer (1) ;
45
46
47
48      int x = r.nextInt (high - low) + low ;
49      int y = r.nextInt (high - low) + low ;
50
51      this.baseHealth = 2 ;
52      this.coordinate = new Vector2f () ;
53      this.coordinate.x = x ;
```

```

54     this.coordinate.y = y;
55     this.isHoldingLeft = false;
56     this.isHoldingRight = false;
57     this.isHoldingUp = false;
58     this.isHoldingDown = false;
59     this.isHoldingPause = false;
60     this.isHoldingUnPause = false;
61     this.skillAlgorithm = new GoingBackInTimeSkill();
62     this.size = 40;
63     this.speed = 5f;
64     this.baseSpeed = 5f;
65     this.health = 2;
66     this.bombCount = 2;
67     //this.playerStats = new ConcretePlayer();
68 }
69
70 public void setBombTimer(int time){
71     bombTimer = 30*time;
72 }
73
74
75 public void reduceTimer(){
76     bombTimer--;
77 }
78
79
80 public void ChangePlayerLocation(int x, int y){
81
82     this.coordinate.x = x;

```

```
83     this.coordinate.y = y;
84
85 }
86
87 public PlayerInfo(int id, Vector2f coordinate)
88 {
89     this.id = id;
90     this.coordinate = coordinate;
91     this.isHoldingLeft = false;
92     this.isHoldingRight = false;
93     this.isHoldingUp = false;
94     this.isHoldingDown = false;
95     this.isHoldingUse = false;
96     this.isHoldingPause = false;
97     this.isHoldingUnPause = false;
98     this.size = 40;
99     this.speed = 2.5f;
100    this.baseSpeed = 2.5f;
101    this.health = 3;
102    this.bombCount = 2;
103    //Test
104 }
105
106 public SkillAlgorithm getSkillAlgorithm()
107 {
108     return skillAlgorithm;
109 }
110
111 public void setSkillAlgorithm(SkillAlgorithm skillAlgorithm)
```

```
112 {
113     this.skillAlgorithm = skillAlgorithm;
114 }
115
116
117 public void onTick()
118 {
119     this.skillAlgorithm.onTick(this);
120 }
121
122 public void tryUsingSpell()
123 {
124     this.skillAlgorithm.useSkill(this);
125 }
126
127 public int getCooldown()
128 {
129     return this.skillAlgorithm.getCooldown();
130 }
131
132 public String getName()
133 {
134     return this.skillAlgorithm.getName();
135 }
136
137 public PacketUpdatePlayerPos getPacketUpdatePlayerPos(int id)
138 {
139     PacketUpdatePlayerPos newPack = new PacketUpdatePlayerPos();
140     newPack.id = id;
```

```

141     newPack.coordinate = this.coordinate ;
142     newPack.isHoldingLeft = this.isHoldingLeft ;
143     newPack.isHoldingRight = this.isHoldingRight ;
144     newPack.isHoldingUp = this.isHoldingUp ;
145     newPack.isHoldingDown = this.isHoldingDown ;
146     newPack.isHoldingUse = this.isHoldingUse ;
147     newPack.size = this.size ;
148     newPack.isHoldingSkill = this.isHoldingSkill ;
149     newPack.isHoldingPause = this.isHoldingPause ;
150     newPack.isHoldingUnPause = this.isHoldingUnPause ;
151     newPack.skillName = this.getName() ;
152     newPack.skillCooldown = this.getCooldown() ;
153     newPack.health = this.health ;
154     return newPack ;
155 }
156
157 public void restoreState(Memento memento){
158
159     if(memento.getState(this)){
160         System.out.println("Successfully restored state");
161     }else{
162         System.out.println("Unable to restore for Caretaker " + this.id);
163     }
164
165 }
166
167 public Memento saveState(){
168     return new Memento(id , coordinate , placedBomb , health , baseHealth ,
    speed , baseSpeed , size , bombCount , bombTimer , deathCounter) ;

```

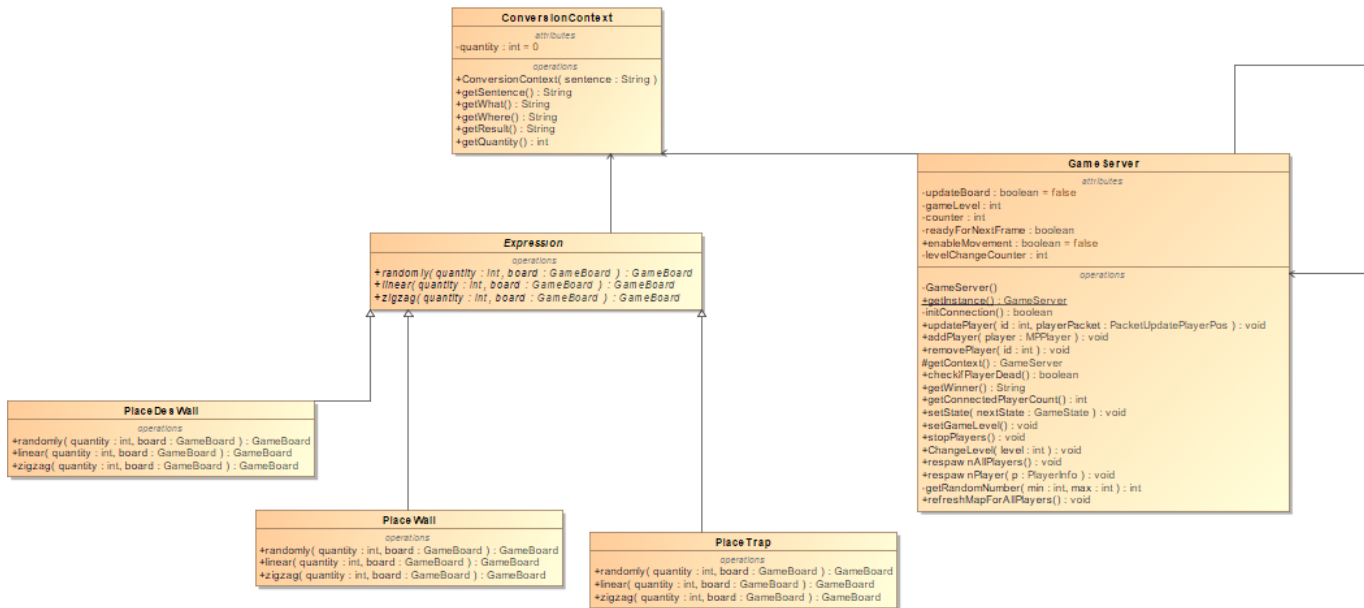
```
169 }
170
171 public int getId () {
172     return id ;
173 }
174
175 public void setId(int id) {
176     this.id = id ;
177 }
178
179 public Vector2f getCoordinate () {
180     return coordinate ;
181 }
182
183 public void setCoordinate (Vector2f coordinate) {
184     this.coordinate = new Vector2f(coordinate.x, coordinate.y);
185 }
186
187 public boolean isPlacedBomb () {
188     return placedBomb ;
189 }
190
191 public void setPlacedBomb (boolean placedBomb) {
192     this.placedBomb = placedBomb ;
193 }
194
195 public int getHealth () {
196     return health ;
197 }
```



```
198
199 public void setHealth (int health) {
200     this.health = health ;
201 }
202
203 public int getBaseHealth () {
204     return baseHealth ;
205 }
206
207 public void setBaseHealth (int baseHealth) {
208     this.baseHealth = baseHealth ;
209 }
210
211 public float getSpeed () {
212     return speed ;
213 }
214
215 public void setSpeed (float speed) {
216     this.speed = speed ;
217 }
218
219 public float getBaseSpeed () {
220     return baseSpeed ;
221 }
222
223 public void setBaseSpeed (float baseSpeed) {
224     this.baseSpeed = baseSpeed ;
225 }
226
```

```
227 public int getSize() {
228     return size;
229 }
230
231 public void setSize(int size) {
232     this.size = size;
233 }
234
235 public int getBombCount() {
236     return bombCount;
237 }
238
239 public void setBombCount(int bombCount) {
240     this.bombCount = bombCount;
241 }
242
243 public int getDeathCounter() {
244     return deathCounter;
245 }
246
247 public void setDeathCounter(int deathCounter) {
248     this.deathCounter = deathCounter;
249 }
250
251 public int getBombTimer() {
252     return bombTimer;
253 }
254
255 }
```

Interpreter



Šis šablonas buno naudingas pagyvinti žaidimą naudojant console komandas

Interpreter

```
1 package server;
2
3 import java.util.Random;
4
5 public class PlaceDesWall extends Expression{
6     @Override
7     public GameBoard randomly(int quantity , GameBoard board) {
8
9         Random r = new Random();
10        int low = 1;
11        int high = 19;
12
13
14        for (int i = 0; i<quantity ; i++){
15            int x = r.nextInt(high -low) + low ;
16            int y = r.nextInt(high -low) + low ;
17
18            board.addObject(x, y, "desWall");
19
20        }
21
22
23        return board;
24    }
25
26    @Override
27    public GameBoard linear(int quantity , GameBoard board) {
```



```

29
30
31     for (int i = 0; i<quantity ; i++){
32
33         int x = 9;
34         int y = 9 + i;
35
36
37
38
39         board.addObject(x, y, "desWall");
40
41     }
42
43
44     return board;
45 }
46
47 @Override
48 public GameBoard zigzag(int quantity , GameBoard board) {
49
50
51     int kiekis = quantity ;
52
53
54
55
56
57     for (int i = 1; i<20; i++){

```

```
58
59     int x = i;
60     int y = i;
61
62
63     if(kiekis != 0){
64
65         kiekis --;
66     }else
67     {
68         board.addObject(x, y, "desWall");
69     }
70
71
72
73 }
74
75
76     return board;
77 }
78 }
```

```
1 package server;
2
3 import java.util.Random;
4
5 public class PlaceWall extends Expression{
6
7
```

```
@Override
```

```
public GameBoard randomly(int quantity , GameBoard board) {
```

```
    Random r = new Random();
```

```
    int low = 1;
```

```
    int high = 19;
```

```
    for (int i = 0; i<quantity; i++){
```

```
        int x = r.nextInt(high - low) + low;
```

```
        int y = r.nextInt(high - low) + low;
```

```
        board.addObject(x, y, "wall");
```

```
    }
```

```
    return board;
```

```
}
```

```
@Override
```

```
public GameBoard linear(int quantity , GameBoard board) {
```



```
37     for (int i = 0; i<quantity ; i++){
38
39         int x = 9 + i;
40         int y = 9;
41
42
43
44
45         board.addObject(x, y, "wall");
46
47     }
48
49
50     return board;
51 }
52
53 @Override
54 public GameBoard zigzag(int quantity , GameBoard board) {
55
56
57     for (int i = 0; i<quantity ; i++){
58
59         int x = 8 + i;
60         int y = 8 + i;
61
62
63
64         board.addObject(x, y, "wall");
65
```

```
66
67     }
68
69
70     return board;
71 }
72 }
```

```
1 package server;
2
3 import java.util.Random;
4
5 public class PlaceTrap extends Expression{
6     @Override
7     public GameBoard randomly(int quantity , GameBoard board) {
8
9         Random r = new Random();
10        int low = 1;
11        int high = 19;
12
13
14
15        for (int i = 0; i<quantity ; i++){
16            int x = r.nextInt(high -low) + low ;
17            int y = r.nextInt(high -low) + low ;
18
19            board.addObject(x, y, "trap");
20
21        }
```

```

22
23
24     return board;
25 }
26
27 @Override
28 public GameBoard linear(int quantity , GameBoard board) {
29
30
31     for (int i = 0; i<quantity ; i++){
32
33         int x = 4 + i*2;
34         int y = 9;
35
36
37
38
39         board.addObject(x, y, "trap");
40
41     }
42
43
44     return board;
45 }
46
47 @Override
48 public GameBoard zigzag(int quantity , GameBoard board) {
49
50     int reverse = -1;

```

```

51
52     for (int i = 0; i<quantity; i++){
53
54         int x = 4 + i;
55         int y = 9 + (i * reverse);
56
57         reverse *= -1;
58
59
60
61         board.addObject(x, y, "trap");
62
63     }
64
65
66     return board;
67 }
68 }

```

```

1 package server;
2
3 public abstract class Expression {
4
5     public abstract GameBoard randomly(int quantity, GameBoard board);
6     public abstract GameBoard linear(int quantity, GameBoard board);
7     public abstract GameBoard zigzag(int quantity, GameBoard board);
8
9
10

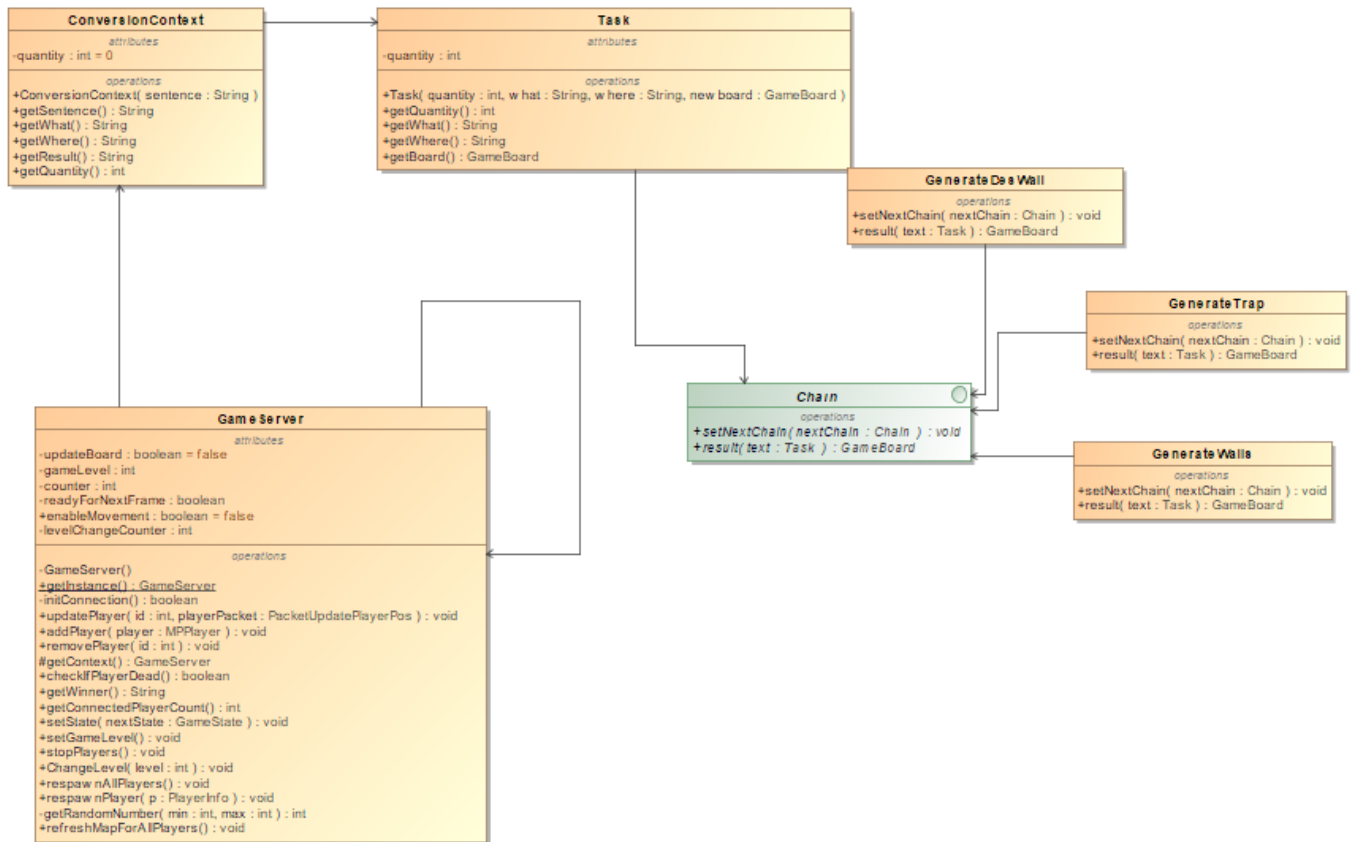
```

11 }

```
1 package server;
2
3 import java.util.Locale;
4
5 public class ConversionContext {
6
7     private String givenSentence = "";
8
9     private String result = "";
10
11     private String what = "";
12     private String where = "";
13
14     private int quantity = 0;
15
16     String[] parts;
17
18     public ConversionContext(String sentence){
19
20         givenSentence = sentence.toLowerCase();
21
22         parts = getSentence().split(" ");
23
24         what = parts[2];
25
26         quantity = Integer.parseInt(parts[1]);
27
```

```
28     where = parts [3];
29
30     result = givenSentence + " ";
31
32 }
33
34 public String getSentence (){
35     return givenSentence ;
36 }
37
38 public String getWhat(){
39     return what;
40 }
41
42 public String getWhere (){
43     return where;
44 }
45
46 public String getResult(){
47     return result;
48 }
49
50 public int getQuantity (){
51     return quantity ;
52 }
53
54
55
56 }
```

Chain of responsibility



Šį šabloną pasirinkau, kad pagyvinti interpreter darbą

Chain of responsibility

```
1 package server;
2
3 public class Task {
4     private int quantity;
5     private GameBoard board;
6     private String what;
7     private String where;
8
9     public Task(int quantity , String what, String where , GameBoard
newboard){
10         this.quantity = quantity;
11         this.what = what;
12         this.where = where;
13         board = newboard;
14     }
15
16
17
18     public int getQuantity(){
19         return quantity;
20     }
21
22     public String getWhat(){
23         return what;
24     }
25
26     public String getWhere(){
```

return where ;

```

28     }
29
30     public GameBoard getBoard () {
31         return board;
32     }
33 }

```

```

1 package server;
2
3 public interface Chain {
4
5     public void setNextChain (Chain nextChain);
6     public GameBoard result (Task text);
7 }

```

```

1 package server;
2
3 public class GenerateWalls implements Chain {
4
5
6     private Chain nextInChain;
7
8     @Override
9     public void setNextChain (Chain nextChain) {
10         nextInChain = nextChain;
11
12     }
13
14     @Override
15     public GameBoard result (Task text) {

```

```

16
17     GameBoard result = text.getBoard();
18
19     if (text.getWhat().equals("wall")){
20
21         PlaceWall newWalls = new PlaceWall();
22
23         System.out.println("Turiu sienas sukurti ");
24
25         switch (text.getWhere()){
26             case "randomly": result = newWalls.randomly(text.
getQuantity(), text.getBoard());
27                 break;
28
29             case "zigzag": result = newWalls.zigzag(text.
getQuantity(), text.getBoard());
30                 break;
31
32             case "linear": result = newWalls.linear(text.
getQuantity(), text.getBoard());
33                 break;
34
35         }
36
37         return result;
38
39
40
41     } else {

```

```
42         nextInChain . result ( text );
43     }
44
45
46
47     return result ;
48
49 }
50 }
```

```
1 package server ;
2
3 public class GenerateTrap implements Chain {
4     private Chain nextInChain ;
5
6     @Override
7     public void setNextChain ( Chain nextChain ) {
8         nextInChain = nextChain ;
9
10    }
11
12    @Override
13    public GameBoard result ( Task text ) {
14
15        GameBoard result = text . getBoard ( ) ;
16
17        if ( text . getWhat ( ) . equals ( " trap " ) ) {
18
19            PlaceTrap newWalls = new PlaceTrap ( ) ;
```

```
20
21     switch (text.getWhere()){
22         case "randomly": result= newWalls.randomly(text.
getQuantity(),text.getBoard());
23         break;
24
25         case "zigzag": result = newWalls.zigzag(text.
getQuantity(),text.getBoard());
26         break;
27
28         case "linear": result= newWalls.linear(text.
getQuantity(),text.getBoard());
29         break;
30
31     }
32
33     return result;
34
35
36
37     }else{
38         nextInChain.result(text);
39     }
40
41
42
43     return result;
44
45 }
```

46

47 }

1 package server;

2

3 public class GenerateDesWall implements Chain {

4 private Chain nextInChain;

5

6 @Override

7 public void setNextChain(Chain nextChain) {

8 nextInChain = nextChain;

9

10 }

11

12 @Override

13 public GameBoard result(Task text) {

14

15 GameBoard result = text.getBoard();

16

17 if (text.getWhat().equals("deswall")){

18

19 PlaceDesWall newWalls = new PlaceDesWall();

20

21 switch (text.getWhere()){

22 case "randomly": result = newWalls.randomly(text.
getQuantity(), text.getBoard());

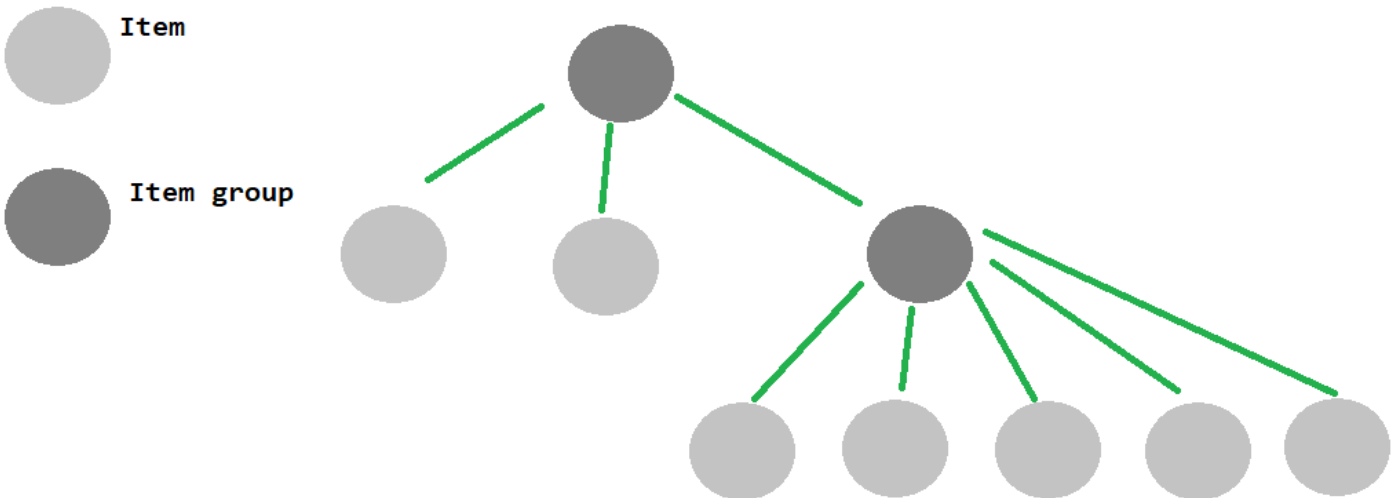
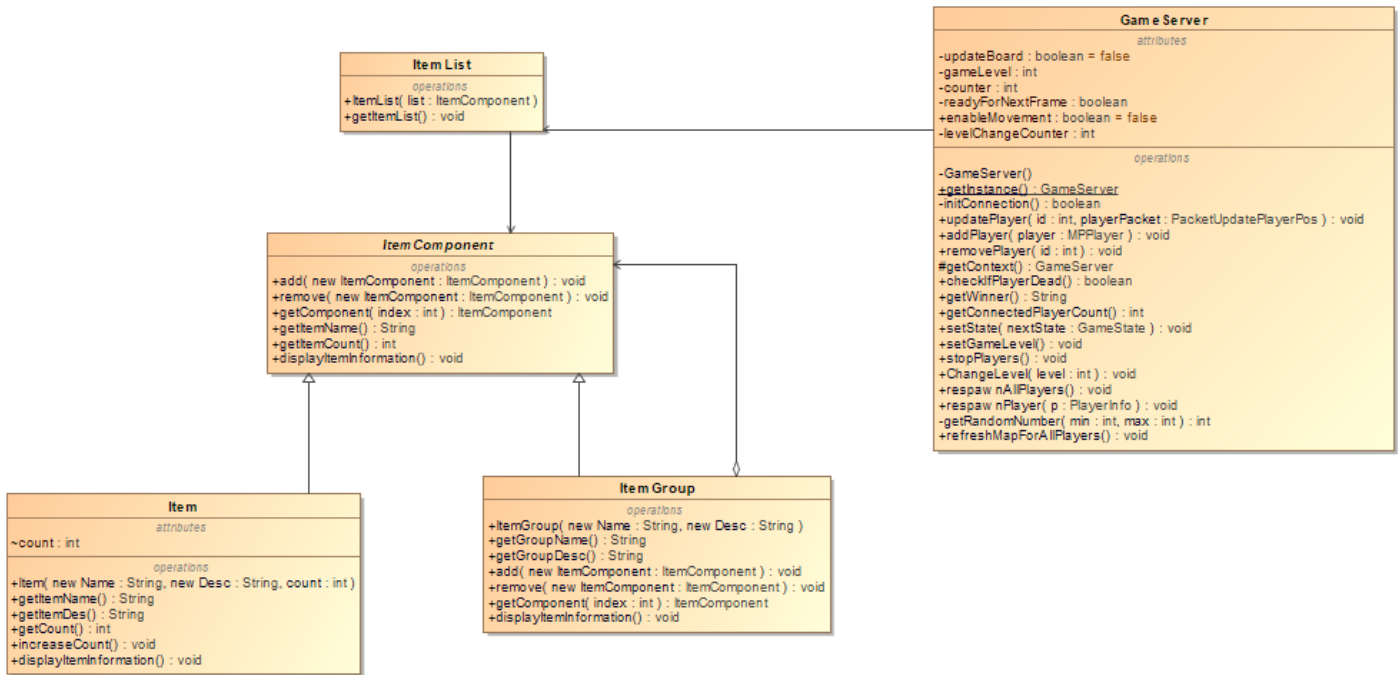
23 break;

24

25 case "zigzag": result = newWalls.zigzag(text.

```
getQuantity () , text . getBoard () );  
26         break ;  
27  
28         case  "linear" : result = new Walls . linear ( text .  
getQuantity () , text . getBoard () );  
29         break ;  
30  
31     }  
32  
33     return  result ;  
34  
35  
36  
37 }  
38  
39  
40  
41     return  result ;  
42  
43 }  
44 }
```

Composite



Šį algoritmą pasirinkau, nes reikėjo būdo surinkti informaciją apie žaidimą ir ją pavaizduoti.

Composition

```
package server;  
  
public class Item extends ItemComponent{  
    String itemName;  
    String itemDes ;  
}
```

1
2
3
4
5
6

```
7      int count ;
8
9      public Item (String newName, String newDesc, int count){
10         itemName = newName;
11         itemDes = newDesc ;
12         this.count = count ;
13     }
14
15
16     public String getItemName () {
17         return itemName ;
18     }
19
20     public String getItemDes () {
21         return itemDes ;
22     }
23
24     public int getCount () {
25         return count ;
26     }
27
28     public void increaseCount () {
29         count++;
30     }
31
32     public void displayItemInformation () {
33         System.out.println (itemName + " Desc: " + itemDes + " Count: "
34         + count);
35     }
```

35
36
37
38
39 }

```
1 package server;
2
3 public abstract class ItemComponent {
4
5     public void add(ItemComponent newItemComponent){
6         throw new UnsupportedOperationException();
7     }
8
9     public void remove(ItemComponent newItemComponent){
10        throw new UnsupportedOperationException();
11    }
12
13    public ItemComponent getComponent(int index){
14        throw new UnsupportedOperationException();
15    }
16
17    public String getItemName(){
18        throw new UnsupportedOperationException();
19    }
20
21    public int getItemCount(){
22        throw new UnsupportedOperationException();
23    }
```

```
24
25     public void displayItemInformation () {
26         throw new UnsupportedOperationException ();
27     }
28
29
30
31 }
```

```
1 package server;
2
3
4 import java . util . ArrayList ;
5 import java . util . Iterator ;
6
7 public class ItemGroup extends ItemComponent {
8
9     ArrayList itemComponents = new ArrayList ();
10
11     String groupName;
12     String groupDesc ;
13
14     public ItemGroup (String newName, String newDesc) {
15         groupName = newName;
16         groupDesc = newDesc ;
17     }
18
19
20     public String getGroupName () {
```

```
21         return groupName ;
22     }
23
24     public String getGroupDesc () {
25         return groupDesc ;
26     }
27
28     public void add (ItemComponent newItemComponent) {
29         itemComponents . add (newItemComponent) ;
30     }
31
32     public void remove (ItemComponent newItemComponent) {
33         itemComponents . remove (newItemComponent) ;
34     }
35
36     public ItemComponent getComponent (int index) {
37
38         return (ItemComponent) itemComponents . get (index) ;
39     }
40
41
42     public void displayItemInformation () {
43         System . out . println (getGroupName () + "\n" + getGroupDesc () + "\n");
44
45
46         Iterator itemIterator = itemComponents . iterator () ;
47
48         while (itemIterator . hasNext ()) {
```

```

49         ItemComponent itemInfo = (ItemComponent) itemIterator.next
        ();
50         itemInfo.displayItemInformation();
51     }
52
53     System.out.println();
54     System.out.println();
55 }
56
57
58
59
60
61 }

```

```

1 package server;
2
3 public class ItemList {
4
5     ItemComponent itemList;
6
7     public ItemList(ItemComponent list){
8         itemList = list;
9     }
10
11     public void getItemList(){
12         itemList.displayItemInformation();
13     }
14

```

